

ESP-IoT-Solution 用户指南



Release master
乐鑫信息科技
2025 年 10 月 11 日

Table of contents

Table of contents	i
1 快速入门	3
1.1 ESP-IoT-Solution 版本说明	3
1.2 开发环境搭建	3
1.3 硬件准备	3
1.4 如何获取和使用组件	4
1.5 如何使用示例程序	4
2 基础组件	7
2.1 通信总线组件 (Bus)	7
2.1.1 i2c_bus	7
2.1.2 spi_bus	14
2.2 I2S LCD 驱动	18
2.2.1 API 参考	18
2.3 板级支持组件 (Boards)	21
2.3.1 使用方法	22
2.3.2 开发板切换和配置	22
2.3.3 已支持的开发板	24
2.3.4 添加新的开发板	25
2.3.5 组件依赖	25
2.3.6 已适配 IDF 版本	25
2.3.7 已适配芯片	25
2.4 CMake Utilities	25
2.4.1 如何使用	25
2.4.2 详细参考	26
3 蓝牙	27
3.1 BLE 连接管理	27
3.1.1 应用示例	27
3.1.2 示例	27
3.1.3 API 参考	27
3.2 BLE 服务	38
3.2.1 警报通知服务	38
3.2.2 电池服务	41
3.2.3 人体组织成分服务	55
3.2.4 当前时间服务	58
3.2.5 设备信息服务	61
3.2.6 心率服务	65
3.2.7 健康温度计服务	68
3.2.8 TX 电源服务	71
3.2.9 用户数据服务	72
3.2.10 体重秤服务	75
3.3 BLE 配置文件	79
3.3.1 警报通知配置文件	79
3.3.2 心率配置文件	83
3.3.3 健康温度计配置文件	86

3.4	BLE HCI 组件	89
3.4.1	BLE HCI 使用方法	89
3.4.2	API 参考	89
4	显示	97
4.1	LCD 显示屏	97
4.1.1	LCD 概述	97
4.1.2	LCD 术语表	108
4.1.3	LCD 开发指南	108
4.1.4	SPI LCD 详解	113
4.1.5	RGB LCD 详解	120
4.1.6	MIPI DSI LCD 详解	133
4.1.7	LCD 屏幕撕裂详解	147
4.1.8	LCD Application Solution	150
4.1.9	GUI 优化解决方案	157
4.2	数码管驱动	160
4.2.1	CH450 驱动	160
4.2.2	HT16C21 驱动	161
4.2.3	IS31FL3XXX 驱动	162
4.3	LED 指示灯	162
4.3.1	支持的指示灯类型	163
4.3.2	定义闪烁类型	163
4.3.3	预定义闪烁优先级	167
4.3.4	控制指示灯闪烁	167
4.3.5	自定义指示灯闪烁	168
4.3.6	gamma 曲线调光	168
4.3.7	驱动电平设置	168
4.3.8	API 参考	168
4.4	LCD 工具	172
4.4.1	ESP LV DECODER	172
4.4.2	ESP LV FS	174
4.4.3	ESP MMAP ASSETS	175
5	USB 主机 & 设备	183
5.1	USB 外设综述	183
5.1.1	ESP USB 外设介绍	183
5.1.2	USB-OTG 外设介绍	186
5.1.3	USB-Serial-JTAG 外设介绍	188
5.1.4	USB PHY/Transceiver 介绍	190
5.1.5	USB VID 和 PID	192
5.1.6	USB Host 方案	193
5.1.7	USB Device 方案	196
5.1.8	自供电 USB 设备解决方案	198
5.1.9	阻止 Windows 依据 USB 设备序列号递增 COM 编号	200
5.1.10	USB 信号质量测试	200
5.1.11	TinyUSB 应用指南	206
5.1.12	使用原生的 tinyusb 进行开发	209
5.2	USB 主机驱动	213
5.2.1	USB Stream 组件说明	213
5.2.2	ESP MSC OTA	224
5.2.3	USB 主机 CDC	228
5.2.4	USB PPP	237
5.2.5	USB RNDIS 主机驱动	238
5.2.6	USB ECM 主机驱动	241
5.3	USB 设备驱动	243
5.3.1	USB Device UVC	243
5.3.2	ESP Device UAC	247
5.3.3	ESP TinyUF2	249

6	音频	255
6.1	PWM 音频	255
6.1.1	特性	255
6.1.2	结构	255
6.1.3	PWM 频率	256
6.1.4	应用示例	256
6.1.5	API 参考	256
6.2	DAC 音频	260
6.2.1	API 参考	260
6.3	ADC 麦克风	262
6.3.1	特性	262
6.3.2	参考电路	262
6.3.3	外部配置 ADC Continuous	262
6.3.4	与 ADC oneshot 模式一起使用	263
6.3.5	示例代码	263
6.3.6	API 参考	263
7	音视频编解码	265
7.1	AVI 播放器	265
7.1.1	API 参考	265
8	图形界面	271
8.1	LVGL 图形库	271
8.1.1	主要特性	271
8.1.2	配置要求	271
8.1.3	在线工具	271
8.1.4	示例方案	272
9	人工智能	275
9.1	OpenAI	275
9.1.1	FAQ	275
9.1.2	API 参考	276
9.2	Touch 手写数字识别	290
9.2.1	Touch 原理与数据采集	290
9.2.2	模型训练与部署	292
9.2.3	端侧推理	296
10	输入设备	297
10.1	按键	297
10.1.1	按键事件	298
10.1.2	配置项	299
10.1.3	应用示例	300
10.1.4	API Reference	304
10.2	键盘扫描	309
10.2.1	组件事件	310
10.2.2	应用示例	310
10.2.3	API Reference	311
10.3	旋钮	315
10.3.1	适用的场景	316
10.3.2	硬件设计	316
10.3.3	旋钮事件	316
10.3.4	配置项	317
10.3.5	应用示例	317
10.3.6	API Reference	318
10.4	触摸屏驱动	322
10.4.1	校准触摸屏	322
10.4.2	触摸屏的按下	322
10.4.3	触摸屏的旋转	323
10.4.4	应用示例	323

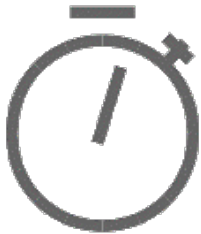
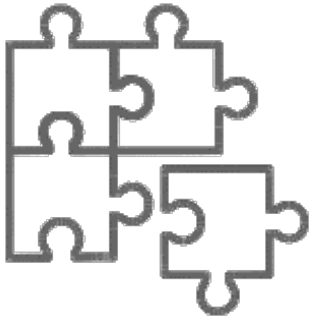
10.4.5	API 参考	324
11	红外	329
11.1	红外学习	329
11.1.1	应用示例	329
11.1.2	API Reference	331
12	低功耗方案	335
12.1	ESP32-P4 备用电池供电方案	335
12.1.1	电池连接与电路保护	335
12.1.2	电源切换	337
12.1.3	欠压检测器 & BOD filter	338
12.1.4	电池充电电路	339
12.1.5	VBAT 供电场景下的外设支持功能	340
13	传感器集	341
13.1	Sensor Hub 简介	341
13.1.1	Sensor Hub 使用方法	341
13.1.2	示例程序	343
13.1.3	API 参考	343
13.2	温湿度传感器	352
13.2.1	已适配列表	352
13.2.2	API 参考	353
13.3	惯性传感器 (IMU)	353
13.3.1	已适配列表	353
13.3.2	API 参考	353
13.4	环境光传感器	353
13.4.1	已适配列表	353
13.4.2	API 参考	353
13.5	气压传感器	354
13.5.1	已适配列表	354
13.6	手势传感器	354
13.6.1	已适配列表	354
13.7	热敏电阻传感器	354
13.7.1	应用示例	355
13.7.2	API 参考	355
13.8	功率监视器	357
13.8.1	适配列表	357
13.8.2	API 参考	357
13.9	功率测量	359
13.9.1	适配列表	359
13.9.2	API 参考	359
13.10	电池电量计	365
13.10.1	已适配列表	365
13.11	ADC 传感器	365
13.11.1	已适配列表	366
14	触摸传感器	367
14.1	触摸接近感应传感器	367
14.1.1	实现原理	367
14.1.2	测试硬件参考	368
14.1.3	配置参考	368
14.1.4	参数调节参考	369
14.1.5	示例程序	374
14.1.6	API Reference	374
14.2	触摸按键传感器	376
14.2.1	概述	376
14.2.2	配置参数	376
14.2.3	API 使用示例	377

14.2.4	示例	378
14.2.5	API 参考	378
14.3	触摸按键传感器	381
14.3.1	概述	381
14.3.2	配置参数	381
14.3.3	API 使用示例	382
14.3.4	示例	384
14.3.5	API 参考	384
15	存储方案	389
15.1	存储媒介	389
15.1.1	SPI Flash	389
15.1.2	SD 卡	390
15.1.3	eMMC	390
15.1.4	EEPROM	391
15.2	文件系统	391
15.2.1	NVS 库	391
15.2.2	FAT 文件系统	392
15.2.3	SPIFFS 文件系统	392
15.2.4	LittleFS 文件系统	393
15.2.5	虚拟文件系统 (VFS)	393
15.2.6	存储安全	393
15.2.7	文件系统设计建议	394
16	电机	395
16.1	无刷电机	395
16.1.1	无刷电机控制概述	395
16.1.2	基于 ADC 采样的无感方波电机控制	399
16.1.3	基于比较器检测的无感方波电机控制	402
16.1.4	ESP Sensorless BLDC Control Components	404
16.2	舵机	414
16.2.1	使用方法	414
16.2.2	应用示例	415
16.2.3	API 参考	415
16.3	ESP SimpleFOC	417
16.3.1	在您的项目中使用 SimpleFOC	417
16.3.2	API 参考	419
17	安全 & 加密	421
17.1	Flash 加密	421
17.1.1	Flash 加密与安全启动	421
17.1.2	加密范围	421
17.1.3	Flash 加密功能支持列表	422
17.1.4	开发模式和发布模式	422
17.1.5	加密过程	422
17.1.6	应用程序访问加密分区	423
17.1.7	重新烧录 Flash	423
17.1.8	取消加密	423
17.1.9	示例代码	423
17.1.10	最佳实践	424
17.1.11	常见问题 (FAQ)	424
17.2	安全启动	424
17.2.1	安全启动与 Flash 加密	424
17.2.2	安全启动信任链	424
17.2.3	安全启动功能支持列表	425
17.2.4	签名与密钥管理	425
17.2.5	签名块格式	425
17.2.6	启用硬件安全启动	426
17.2.7	密钥撤销	426

17.2.8	启用后的限制	427
17.2.9	软件签名验证	427
17.2.10	示例代码	428
17.2.11	最佳实践	428
17.2.12	常见问题 (FAQ)	428
17.3	量产方案	428
17.3.1	Flash 下载工具	428
17.3.2	乐鑫产测工具	429
17.3.3	模组治具制作规范	430
17.3.4	Matter QR 二维码生成工具	431
17.3.5	EspRFTTestTool 工具包	432
18	电工 & 照明方案	435
18.1	球泡灯驱动	435
18.1.1	调光方案	435
18.1.2	渐变原理	438
18.1.3	低功耗实现流程	438
18.1.4	颜色校准方案	438
18.1.5	功率限制方案	439
18.1.6	API 参考	440
19	解决方案	457
19.1	4G 解决方案	457
19.1.1	IOT ETH 以太网	457
20	其它资源	461
20.1	GPIO 扩展	461
20.1.1	已适配列表	461
20.2	ADC 扩展量程方案	461
20.2.1	Patch 使用方法	462
20.2.2	API 使用说明	462
20.2.3	ADC 量程扩展效果对比	463
20.3	过零检测	464
20.3.1	过零检测事件	464
20.3.2	配置项	464
20.3.3	应用示例	464
20.3.4	API Reference	465
20.4	ADC 两点校准方案	470
20.4.1	ESP32 两点校准原理	470
20.4.2	ESP32-S2 两点校准（方法二）原理	471
20.4.3	API 参考	471
21	Contributions Guide	473
21.1	How to Contribute	473
21.2	Before Contributing	473
21.3	Pull Request Process	473
21.4	Legal Part	473
21.5	Related Documents	474
21.5.1	esp-iot-solution 编码规范	474
索引		481
索引		481

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。

		
入门指南	显示设备	USB 主机 & 设备
		
图形界面	输入设备	传感器集
		
音频设备	安全 & 加密	贡献代码

Chapter 1

快速入门

本文档简要介绍如何获取和使用 ESP-IoT-Solution 内的组件，如何编译和运行示例，帮助初学者快速上手。

1.1 ESP-IoT-Solution 版本说明

ESP-IoT-Solution 自 `release/v2.0` 起采用组件化管理，各组件与示例独立迭代，依赖的 ESP-IDF 版本请查阅组件 `idf_component.yml` 文件。`release` 分支仅维护历史大版本，`master` 分支持续集成新特性。建议新项目通过组件注册表获取所需组件。

不同版本说明如下，详情及组件列表请见 [README_CN.md](#)：

表 1: ESP-IoT-Solution 版本支持情况

ESP-IoT-Solution	依赖的 ESP-IDF	主要变更	支持状态
master	>= v5.3	新芯片支持	新功能开发分支
release/v2.0	<= v5.3, >= v4.4	支持组件管理器	历史版本维护
release/v1.1	v4.0.1	IDF 版本更新，代码迁移	备份，停止维护
release/v1.0	v3.2.2	历史版本	备份，停止维护

1.2 开发环境搭建

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：请参考：[ESP-IDF 编程指南](#) 完成 ESP-IDF 开发环境的搭建。

1.3 硬件准备

您可以选择任意 ESP 系列开发板，或参考 [esp-bsp](#) 中支持的开发板快速开始。各系列芯片规格请见 [ESP 产品选型工具](#)。

ESP 系列 SoC 支持以下功能：

- Wi-Fi（2.4 GHz/5 GHz 双频）

- 蓝牙 5.x (BLE/Mesh)
- 高性能多核处理器，主频最高可达 400 MHz
- 超低功耗协处理器和深度睡眠模式
- 丰富的外设接口：
 - 通用接口：GPIO、UART、I2C、I2S、SPI、SDIO、USB OTG 等
 - 专用接口：LCD、摄像头、以太网、CAN、Touch、LED PWM、温度传感器等
- 大容量内存：
 - 内部 RAM 最大可达 768 KB
 - 支持外部 PSRAM 扩展
 - 支持外部 Flash 存储
- 增强的安全特性：
 - 硬件加密引擎
 - 安全启动
 - Flash 加密
 - 数字签名

ESP 系列 SoC 采用先进工艺制程，提供业界领先的射频性能、低功耗特性和稳定可靠性，适用于物联网、工业控制、智能家居、可穿戴设备等多种应用场景。

备注：各系列芯片的具体规格和功能请参考 [ESP 产品选型工具](#)。

1.4 如何获取和使用组件

推荐通过 [ESP Component Registry](#) 获取 ESP-IoT-Solution 组件。

以 button 组件为例，添加依赖的步骤如下：

1. 在项目根目录下执行：

```
idf.py add-dependency "espressif/button"
```

2. 在代码中引用头文件并调用 API，例如：

```
#include "iot_button.h"
// 具体 API 使用请参考组件文档
```

更多组件管理器的使用方法请参考 [ESP Registry Docs](#)。

1.5 如何使用示例程序

ESP-IoT-Solution 提供了丰富的示例程序，帮助用户快速上手。以 button_power_save 示例为例：

1. 确保已经完成 ESP-IDF 开发环境的搭建，并成功配置了环境变量
2. 下载 ESP-IoT-Solution 代码仓库：

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

3. 进入示例目录或将其复制到您的工作目录：

```
cd examples/get-started/button_power_save
```

备注：如果您将示例复制到其他目录，由于文件路径发生变更，请删除 main/idf_component.yml 中所有 override_path 的配置。

4. 选择目标芯片（如 ESP32，首次使用或切换芯片时需执行）：

```
idf.py set-target esp32
```

5. 配置项目（可选）:

```
idf.py menuconfig
```

6. 编译并烧录到开发板:

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

7. 通过串口监视输出:

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

更多示例请见 `examples/` 目录，具体使用方法请参考各示例下的 `README` 文件。

Chapter 2

基础组件

2.1 通信总线组件 (Bus)

2.1.1 i2c_bus

i2c_bus 的使用方法:

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。
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,
```

(下页继续)

(续上页)

```

    .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。
3. 对于 I2C 端口不足或者需要软件 I2C 调试的场景下，可以使用 menuconfig 在 (Top) → Component config → Bus Options → I2C Bus Options 中开启 Enable software I2C support，并在 `i2c_bus_create()` 的 port 中传入 `i2c_sw_port_t` 类型的 port，例如 `I2C_NUM_SW_0`。

已适配 IDF 版本

- ESP-IDF v4.0 及以上版本。

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. Please note that enabling `I2C_BUS_SUPPORT_SOFTWARE` in menuconfig allows you to use ports in `i2c_sw_port_t` to enable software I2C.
- **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

`i2c_master_bus_handle_t i2c_bus_get_internal_bus_handle (i2c_bus_handle_t bus_handle)`

Get internal idf bus_handle from i2c_bus_handle.

参数 **bus_handle** –I2C bus handle

返回 `i2c_master_bus_handle_t` Return the idf bus_handle if obtained successfully, return NULL if failed.

`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.

Structures

struct **i2c_config_t**

I2C initialization parameters.

Public Members

i2c_mode_t **mode**

I2C mode

int **sda_io_num**

GPIO number for I2C sda signal

int **scl_io_num**

GPIO number for I2C scl signal

bool **sda_pullup_en**

Internal GPIO pull mode for I2C sda signal

bool **scl_pullup_en**

Internal GPIO pull mode for I2C scl signal

uint32_t **clk_speed**

I2C clock frequency for master mode, (no higher than 1MHz for now)

struct *i2c_config_t*::[anonymous] **master**

I2C master config

uint32_t **clk_flags**

Bitwise of I2C_SCLK_SRC_FLAG_**FOR_DFS** for clk source choice

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_MEM_16BIT_ADDR

set 16bit mem_address to NULL_I2C_MEM_16BIT_ADDR if i2c device has no internal address during read/write

`NULL_I2C_DEV_ADDR`

invalid i2c device address

`gpio_pad_select_gpio`

`portTICK_RATE_MS`

Type Definitions

`typedef void *i2c_bus_handle_t`

i2c bus handle

`typedef void *i2c_bus_device_handle_t`

i2c device handle

`typedef void *i2c_cmd_handle_t`

I2C command handle

2.1.2 spi_bus

spi_bus 的使用方法

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
```

(下页继续)

(续上页)

```

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

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。

已适配 IDF 版本

- ESP-IDF v4.0 及以上版本。

API 参考

Header File

- `components/spi_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

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。

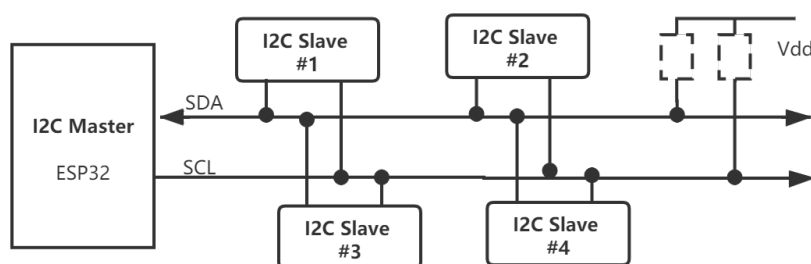


图 1: i2c_bus 连接框图

2.2 I2S LCD 驱动

2.2.1 API 参考

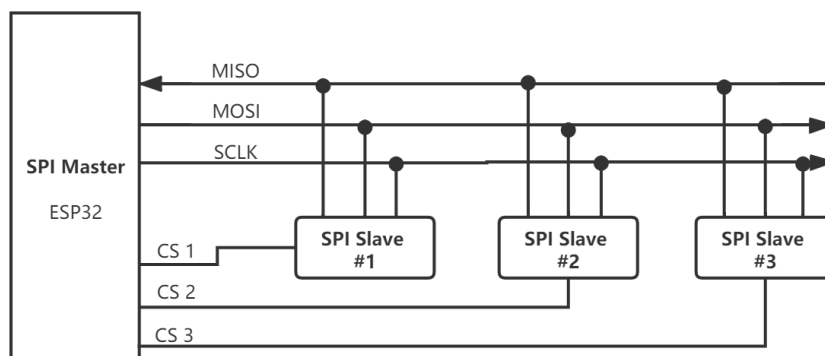


图 2: spi_bus 连接框图

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 deinitilize

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：

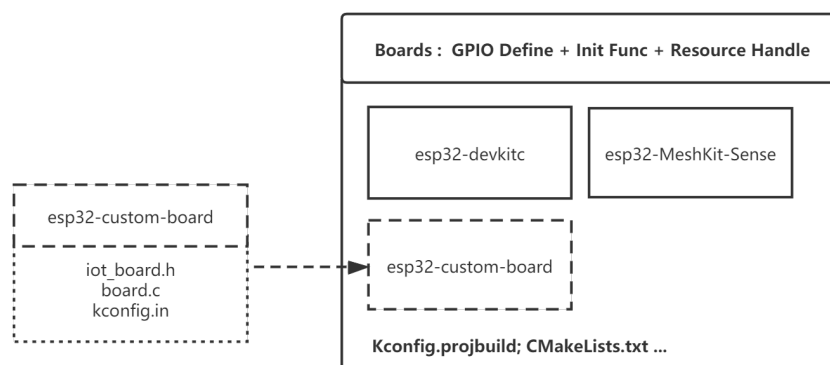


图 3: Boards 组件结构框图

- 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

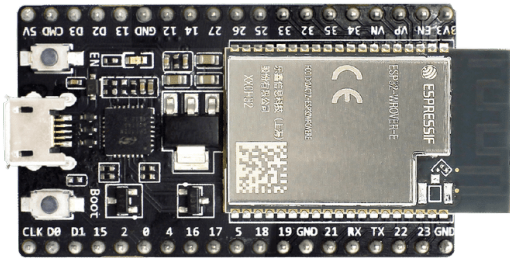
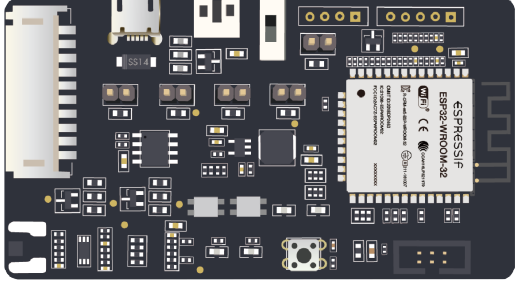
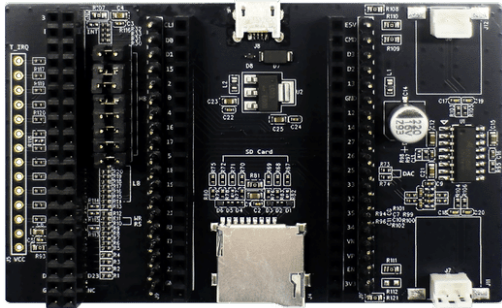
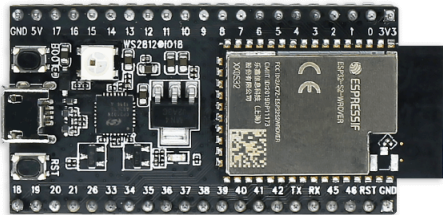
基于 Boards 开发的应用程序，可以使用以下方法切换和配置开发板：

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	
	
esp32-devkitc	esp32-meshkit-sense
	
esp32-lcdkit	
ESP32-S2 开发板	
	
esp32s2-saola	
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

2.4 CMake Utilities

`cmake_utilities` 是 `esp-iot-solution` 组件中常用的 **CMake** 工具集合。

支持的功能：

- `project_include.cmake`: 为项目添加额外的功能，例如 `DIAGNOSTICS_COLOR`。文件将被自动解析，详情请参考 [project-include-cmake](#)。
- `package_manager.cmake`: 提供了管理组件版本等功能。
- `gcc.cmake`: 通过 `menuconfig` 管理 GCC 编译器选项，例如 LTO。
- `relinker.cmake` 提供了一种将 `IRAM` 函数移动到 `Flash` 以节省 `RAM` 空间的方法。
- `gen_compressed_ota.cmake`: 添加了新命令 `idf.py gen_compressed_ota` 以生成 `xz` 压缩的 OTA 二进制文件。详情请参考 [xz](#)。
- `gen_single_bin.cmake`: 添加了新命令 `idf.py gen_single_bin` 以生成单个组合 `bin` 文件（组合 `app`、`bootloader`、分区表等）。

2.4.1 如何使用

1. 在你的组件或项目的 `idf_component.yml` 中添加此组件的依赖。

```
dependencies:
  espressif/cmake_utilities: "0.*"
```

2. 在你的组件或项目的 CMakeLists.txt 中的 idf_component_register 后包含 CMake 功能/脚本。

```
// 注意：使用 include() 时应删除 .cmake 后缀，否则将找不到请求的文件
// 注意：应在 `idf_component_register` 函数之后放置此行
// 只包含你需要的功能。
include(package_manager)
```

3. 然后你可以使用 CMake 脚本提供的相应功能。

2.4.2 详细参考

1. [relinker](#)
2. [gen_compressed_ota](#)
3. [GCC Optimization](#)

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

uint16_t **adv_uuid16**

NOTE: Only Support one 16-bit UUID

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 rssi`

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

`ESP_IOT_ATT_SUCCESS`

Success code and error codes

`ESP_IOT_ATT_INVALID_HANDLE`

`ESP_IOT_ATT_READ_NOT_PERMIT`

`ESP_IOT_ATT_WRITE_NOT_PERMIT`

`ESP_IOT_ATT_INVALID_PDU`

`ESP_IOT_ATT_INSUF_AUTHENTICATION`

`ESP_IOT_ATT_REQ_NOT_SUPPORTED`

`ESP_IOT_ATT_INVALID_OFFSET`

`ESP_IOT_ATT_INSUF_AUTHORIZATION`

ESP_IOT_ATT_PREPARE_Q_FULL

ESP_IOT_ATT_NOT_FOUND

ESP_IOT_ATT_NOT_LONG

ESP_IOT_ATT_INSUF_KEY_SIZE

ESP_IOT_ATT_INVALID_ATTR_LEN

ESP_IOT_ATT_ERR_UNLIKELY

ESP_IOT_ATT_INSUF_ENCRYPTION

ESP_IOT_ATT_UNSUPPORT_GRP_TYPE

ESP_IOT_ATT_INSUF_RESOURCE

ESP_IOT_ATT_DATABASE_OUT_OF_SYNC

ESP_IOT_ATT_VALUE_NOT_ALLOWED

ESP_IOT_ATT_NO_RESOURCES

Application error codes

ESP_IOT_ATT_INTERNAL_ERROR

ESP_IOT_ATT_WRONG_STATE

ESP_IOT_ATT_DB_FULL

ESP_IOT_ATT_BUSY

ESP_IOT_ATT_ERROR

ESP_IOT_ATT_CMD_STARTED

ESP_IOT_ATT_ILLEGAL_PARAMETER

ESP_IOT_ATT_PENDING

ESP_IOT_ATT_AUTH_FAIL

ESP_IOT_ATT_MORE

ESP_IOT_ATT_INVALID_CFG

ESP_IOT_ATT_SERVICE_STARTED

ESP_IOT_ATT_ENCRYPED_MITM

ESP_IOT_ATT_ENCRYPED_NO_MITM

ESP_IOT_ATT_NOT_ENCRYPTED

ESP_IOT_ATT_CONGESTED

ESP_IOT_ATT_DUP_REG

ESP_IOT_ATT_ALREADY_OPEN

ESP_IOT_ATT_CANCEL

ESP_IOT_ATT_STACK_RSP

Common profile and service error codes

ESP_IOT_ATT_APP_RSP

ESP_IOT_ATT_UNKNOWN_ERROR

ESP_IOT_ATT_CCC_CFG_ERR

ESP_IOT_ATT_PRC_IN_PROGRESS

ESP_IOT_ATT_OUT_OF_RANGE

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, uint8_t *att_status)
```

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

Param att_status [in] The attribute return status

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_ble_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_AVAILALBE_ENERGY

BAS_CHR_ENERGY_STATUS_FLAGS_BM_AVAILALBE_BATTERY_CAPACITY

BAS_CHR_ENERGY_STATUS_FLAGS_BM_CHARGE_RATE

BAS_CHR_ENERGY_STATUS_FLAGS_BM_AVAILALBE_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 人体组织成分服务

人体成分服务提供了用于健康以及健身领域的相关数据。

示例

[bluetooth/ble_services/ble_bcs.](#)

API 参考

Header File

- [components/bluetooth/ble_services/bcs/include/esp_bcs.h](#)

Functions

`esp_err_t esp_ble_bcs_set_measurement(esp_bcs_val_t *in_val, bool need_send)`

Set the Body Composition value.

参数

- **in_val** –[in] The pointer to store the Body Composition Information.
- **need_send** –[in] If set to true, the Body Composition Info will send to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

`esp_err_t esp_ble_bcs_init(void)`

Initialization Body Composition Service.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

Structures

`struct esp_bcs_val_t`

Body Composition Measurement Characteristic.

Public Members**uint32_t bcs_flag**

Body Composition flag field

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_bcs_val_t*::[anonymous] timestamp

The date and time

uint8_t user_id

User ID field

uint16_t basal_metabolism

Basal Metabolism field

uint16_t muscle_percentage

Muscle Percentage field

uint16_t muscle_mass

Muscle Mass field

uint16_t fat_free_mass

Fat Free Mass field

uint16_t soft_lean_mass

Soft Lean Mass field

uint16_t body_water_mass

Body Water Mass field

uint8_t impedance

Impedance field

uint16_t **weight**

Weight field

uint16_t **height**

Height field

Macros

BLE_BCS_MAX_VAL_LEN

BLE_BCS_UUID16

BLE_BCS_CHR_UUID16_FEATURE

BLE_BCS_CHR_UUID16_MEASUREMENT

BLE_BCS_FEAT_TIME_STAMP

BLE_BCS_FEAT_MULTI_USER

BLE_BCS_FEAT_BASAL_METABOLISM

BLE_BCS_FEAT_MUSCLE_PERCENTAGE

BLE_BCS_FEAT_MUSCLE_MASS

BLE_BCS_FEAT_FAT_FREE_MASS

BLE_BCS_FEAT_SOFT_LEAN_MASS

BLE_BCS_FEAT_BODY_WATER_MASS

BLE_BCS_FEAT_IMPEDENCE

BLE_BCS_FEAT_WEIGHT

BLE_BCS_FEAT_HEIGHT

BLE_BCS_FEAT_MASS_MEASUREMENT_RESOLUTION

BLE_BCS_FEAT_HEIGHT_RESOLUTION

BLE_BCS_FLAG_MEASUREMENT_UNITS

BLE_BCS_FLAG_TIME_STAMP

BLE_BCS_FLAG_MULTI_USER

BLE_BCS_FLAG_BASAL_METABOLISM

BLE_BCS_FLAG_MUSCLE_PERCENTAGE

BLE_BCS_FLAG_MUSCLE_MASS

BLE_BCS_FLAG_FAT_FREE_MASS

BLE_BCS_FLAG_SOFT_LEAN_MASS

BLE_BCS_FLAG_BODY_WATER_MASS

BLE_BCS_FLAG_IMPEDENCE

BLE_BCS_FLAG_WEIGHT

BLE_BCS_FLAG_HEIGHT

BLE_BCS_FLAG_MULTIPLE_PACKET

3.2.4 当前时间服务

当前时间服务定义了蓝牙设备如何向其他蓝牙设备展示日期和时间信息。

示例

[bluetooth/ble_services/ble_cts](#).

API 参考

Header File

- [components/bluetooth/ble_services/cts/include/esp_cts.h](#)

Functions

`esp_err_t esp_ble_cts_get_current_time(esp_ble_cts_cur_time_t *out_val)`

Read the value of Current Time characteristic.

参数 `out_val` –[in] The pointer to store the Current Time Increment.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

esp_err_t **esp_ble_cts_set_current_time** (*esp_ble_cts_cur_time_t* *in_val, bool need_send)

Set the Current Time characteristic value.

参数

- **in_val** –[in] The pointer to store the Current Time.
- **need_send** –[in] send the current time to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

esp_err_t **esp_ble_cts_get_local_time** (*esp_ble_cts_local_time_t* *out_val)

Read the value of Local Time characteristic.

参数 **out_val** –[in] The pointer to store the Local Time Increment.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

esp_err_t **esp_ble_cts_set_local_time** (*esp_ble_cts_local_time_t* *in_val)

Set the Local Time characteristic value.

参数 **in_val** –[in] The pointer to store the Local Time.

返回

- ESP_OK on successful

esp_err_t **esp_ble_cts_get_reference_time** (*esp_ble_cts_ref_time_t* *out_val)

Read the value of Reference Time characteristic.

参数 **out_val** –[in] The pointer to store the Reference Time Increment.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

esp_err_t **esp_ble_cts_set_reference_time** (*esp_ble_cts_ref_time_t* *in_val)

Set the Reference Time characteristic value.

参数 **in_val** –[in] The pointer to store the Reference Time.

返回

- ESP_OK on successful

esp_err_t **esp_ble_cts_init** (void)

Initialization Current Time Service.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

Structures

struct **esp_ble_cts_cur_time_t**

Current Time.

Public Members

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_cts_cur_time_t*::[anonymous] **timestamp**

The date and time

uint8_t day_of_week

Day_of_week. valid range : 1(Monday) - 7(Sunday), 0 means day of week not known

uint8_t fractions_256

Fractions_256. the number of 1 / 256 fractions of second, valid range : 0 - 255

uint8_t adjust_reason

This field represents reason(s) for adjusting time

struct **esp_ble_cts_local_time_t**

Local Time Information.

Public Members

int8_t timezone

Time zone

uint8_t dst_offset

dst_offset. allowed values : 0, 2, 4, 8, 255 Dst offset

struct **esp_ble_cts_ref_time_t**

reference time information

Public Members

uint8_t time_source

time_source. valid range : 0 - 253 255 means not known Time source

uint8_t time_accuracy

Time accuracy

uint8_t **days_since_update**

days_since_update. valid range : 0 - 254 A value of 255 is used when the time span is greater than or equal to 255 days Days since update

uint8_t **hours_since_update**

hours_since_update. valid range : 0 - 23 A value of 255 is used when the time span is greater than or equal to 255 days Hours since update

Macros

BLE_CTS_UUID16

BLE_CTS_CHR_UUID16_CURRENT_TIME

BLE_CTS_CHR_UUID16_LOCAL_TIME

BLE_CTS_CHR_UUID16_REFERENCE_TIME

BLE_CTS_MANUAL_TIME_UPDATE_MASK

BLE_CTS_EXTERNAL_REFERENCE_TIME_UPDATE_MASK

BLE_CTS_CHANGE_OF_TIME_ZONE_MASK

BLE_CTS_CHANGE_OF_DST_MASK

3.2.5 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

3.2.6 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

3.2.7 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

3.2.8 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`

3.2.9 用户数据服务

用户数据服务用于在体育健身、家庭或医疗环境中展示用户相关数据。这允许客户端远程访问和更新用户数据，以及服务器和客户端之间用户数据的同步。

示例

[bluetooth/ble_services/ble_uds](#).

API 参考

Header File

- [components/bluetooth/ble_services/uds/include/esp_uds.h](#)

Functions

`esp_err_t esp_ble_uds_get_db_incre` ([esp_ble_uds_db_t](#) *out_val)

Read the value of Database Change Increment characteristic.

参数 `out_val` –[in] The pointer to store the Database Change Increment.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

`esp_err_t esp_ble_uds_set_db_incre` ([esp_ble_uds_db_t](#) *in_val, bool need_send)

Set the Database Change Increment characteristic value.

参数

- `in_val` –[in] The pointer to store the Database Change Increment Information.

- **need_send** –[in] If set to true, the User Data Info will send to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

esp_err_t **esp_ble_uds_get_user_ctrl** (*esp_ble_uds_user_ctrl_t* *out_val)

Read the value of User Control characteristic.

参数 **out_val** –[in] The pointer to store the User Control value.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

esp_err_t **esp_ble_uds_set_user_ctrl** (*esp_ble_uds_user_ctrl_t* *in_val, bool need_send)

Set the User Control Point characteristic value.

参数

- **in_val** –[in] The pointer to store the User Control Value.
- **need_send** –[in] If set to true, the User Control Value will send to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

esp_err_t **esp_ble_uds_get_reg_user** (*esp_ble_uds_reg_user_t* *out_val)

Read the value of Register User characteristic.

参数 **out_val** –[in] The pointer to store the Register User value.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

esp_err_t **esp_ble_uds_set_reg_user** (*esp_ble_uds_reg_user_t* *in_val, bool need_send)

Set the Register User characteristic value.

参数

- **in_val** –[in] The pointer to store the Register User Value.
- **need_send** –[in] If set to true, the Register User Value will send to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

esp_err_t **esp_ble_uds_init** (void)

Initialization User Data Service.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

Structures

struct **esp_ble_uds_db_t**

Database Change Increment.

Public Members

uint8_t **len**

Database buffer length

uint8_t **db_buf**[BLE_UDS_ATT_VAL_LEN]

Database buffer

struct **esp_ble_uds_reg_user_t**

Register User Characteristic.

Public Members

uint8_t **first_segment**

0: False, 1: True

uint8_t **last_segment**

0: False, 1: True

uint8_t **rolling_segment_number**

When the Rolling segment number is equal to 63, the value is reset to 0 the next time it is incremented

struct *esp_ble_uds_reg_user_t*::[anonymous] **header**

The structure of the Segmentation Header field

uint8_t **user_name_present**

0: False, 1: True

uint8_t **user_name_truncated**

0: False, 1: True

struct *esp_ble_uds_reg_user_t*::[anonymous] **flag**

Defines the value of the flags field that a server shall use when the register user characteristic is indicate

uint8_t **user_index**

User ID Index

uint8_t **user_name**[BLE_UDS_ATT_VAL_LEN]

User name

struct **esp_ble_uds_user_ctrl_t**

User Control Point.

Public Members

uint8_t **op_code**

User control point opcode

uint8_t **param_len**

Parameter buffer length

uint8_t **parameter**[BLE_UDS_USER_CTRL_POINT_PARAM_LEN]

Parameter depend on opcode

Macros

BLE_UDS_ATT_VAL_LEN

Attribute value length

BLE_UDS_USER_CTRL_POINT_PARAM_LEN

User control point param length

BLE_UDS_UUID16

BLE_UDS_CHR_UUID16_DATABASH_CHG

BLE_UDS_CHR_UUID16_USER_INDEX

BLE_UDS_CHR_UUID16_USER_CTRL

BLE_UDS_CHR_UUID16_REG_USER

BLE_UDS_USER_CTRL_REG_NEW_OP

BLE_UDS_USER_CTRL_CONSENT_OP

BLE_UDS_USER_CTRL_DEL_DATA_OP

BLE_UDS_USER_CTRL_LIST_ALL_OP

BLE_UDS_USER_CTRL_DEL_USER_OP

BLE_UDS_USER_CTRL_RSP_OP

3.2.10 体重秤服务

体重秤服务用于从健康和体育健身的体重秤中获取体重相关数据。

示例

[bluetooth/ble_services/ble_wss](#).

API 参考

Header File

- [components/bluetooth/ble_services/wss/include/esp_wss.h](#)

Functions

`esp_err_t esp_ble_wss_get_measurement (esp_ble_wss_measurement_t *out_val)`

Read the weight measurement characteristic value.

参数 `out_val` –[in] The pointer to store the weight measurement value.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong parameter

`esp_err_t esp_ble_wss_set_measurement (esp_ble_wss_measurement_t *in_val, bool need_send)`

Set the weight measurement characteristic value.

参数

- `in_val` –[in] The pointer to store the weight measurement.
- `need_send` –[in] send the weight measurement information to remote client.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

`esp_err_t esp_ble_wss_init (void)`

Initialization Weight Scale Service.

返回

- ESP_OK on successful
- ESP_ERR_INVALID_ARG on wrong initialization
- ESP_FAIL on error

Structures

struct `esp_ble_wss_feature_t`

Weight Scale Feature.

Public Members

`uint32_t timestamp`

0: Don't Support, 1: Support

`uint32_t user_id`

0: Don't Support, 1: Support

`uint32_t bmi`

0: Don't Support, 1: Support

`uint32_t weight`

0: Don't Support, 1: Support

`uint32_t w_resolution`

If weight support, this filed should present

`uint32_t height`

0: Don't Support, 1: Support

`uint32_t h_resolution`

If height support, this filed should present

struct **esp_ble_wss_measurement_t**
weight Measurement

Public Members

uint32_t **measurement_unit**
0: Kg & meter, 1: reference to weight and height resolution

uint32_t **time_present**
0: Don't contain time information, 1: time stamp present

uint32_t **user_present**
0: Don't contain user index, 1: contain user index

uint32_t **bmi_height_present**
0: Don't contain BMI and height, 1: contain BMI and height

struct *esp_ble_wss_measurement_t*::[anonymous] **flag**
Flag

uint16_t **weight**
weight

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_wss_measurement_t*::[anonymous] **timestamp**
The date and time

uint8_t **user_id**
User index

`uint8_t bmi`

BMI

`uint16_t height`

Height

`uint8_t weight_resolution`

Weight resolution

`uint8_t height_resolution`

Height resolution

Macros

`BLE_WSS_UUID16`

`BLE_WSS_CHR_UUID16_WEIGHT_FEATURE`

`BLE_WSS_CHR_UUID16_WEIGHT_MEASUREMENT`

`BLE_WSS_TIMESTAMP_MASK`

`BLE_WSS_MULTI_USER_MASK`

`BLE_WSS_BMI_MASK`

`BLE_WSS_WEIGHT_RESOLUTION_OFFSET`

`BLE_WSS_WEIGHT_RESOLUTION_NONE`

`BLE_WSS_WEIGHT_RESOLUTION_0P5_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P2_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P1_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P05_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P02_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P01_KG`

`BLE_WSS_WEIGHT_RESOLUTION_0P005_KG`

`BLE_WSS_HEIGHT_RESOLUTION_OFFSET`

`BLE_WSS_HEIGHT_RESOLUTION_NONE`

`BLE_WSS_HEIGHT_RESOLUTION_0P01_M`

`BLE_WSS_HEIGHT_RESOLUTION_0P005_M`

`BLE_WSS_HEIGHT_RESOLUTION_0P001_M`

`BLE_WSS_MEASUREMENT_UINTS_FLAG`

`BLE_WSS_TIME_STAMP_FLAG`

`BLE_WSS_USER_ID_FLAG`

`BLE_WSS_BMI_FLAG`

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_htp_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_htp_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。

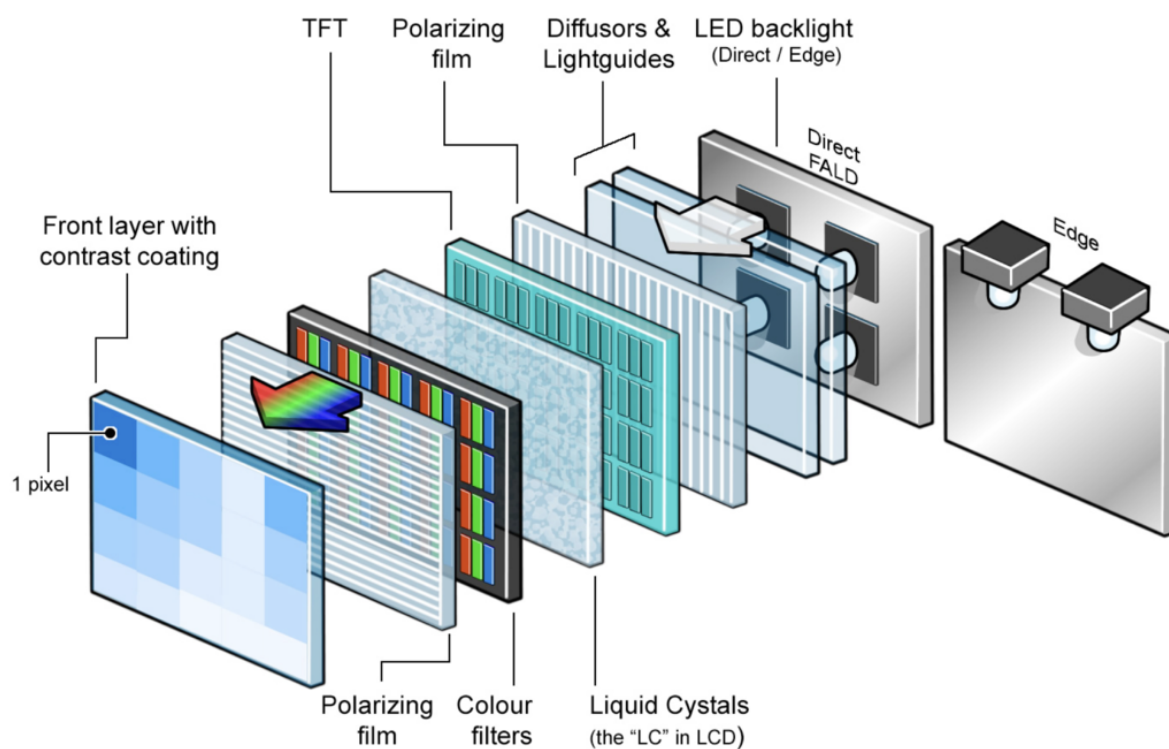


图 1: TFT-LCD 的硬件框图

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

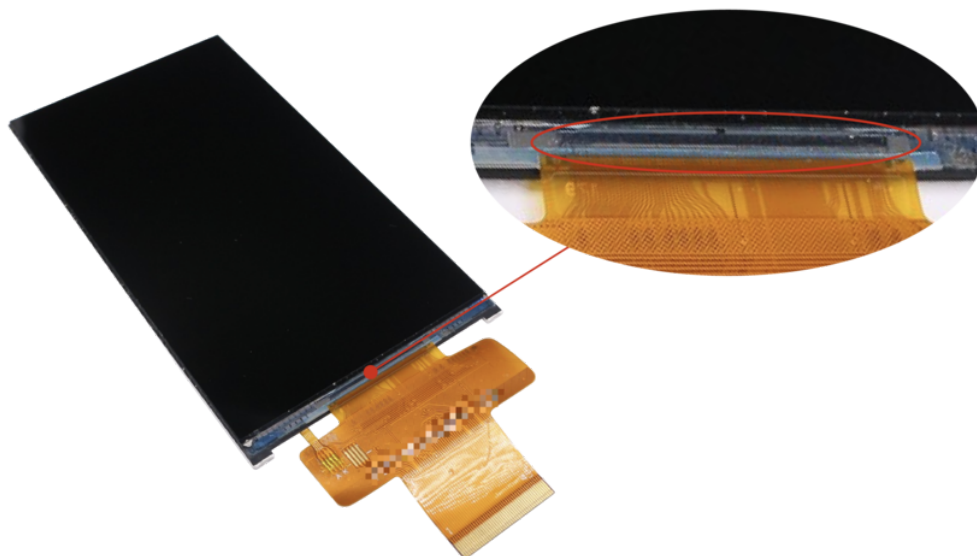


图 2: LCD 模块的驱动 IC



图 3: LCD 矩形屏



图 4: LCD 圆形屏

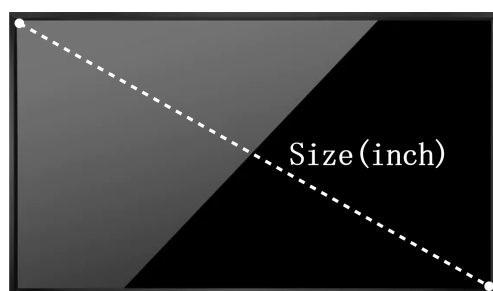


图 5: 屏幕尺寸

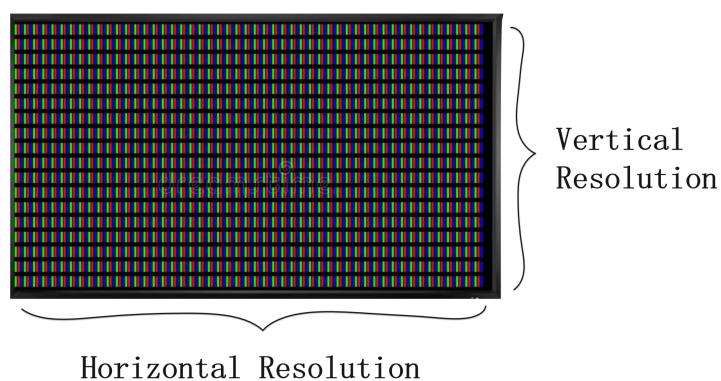


图 6: 屏幕分辨率

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	较大	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：

- SPI 接口的数据传输带宽小，比较适用于低分辨率的屏幕。
- 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。
- 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。
- 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：

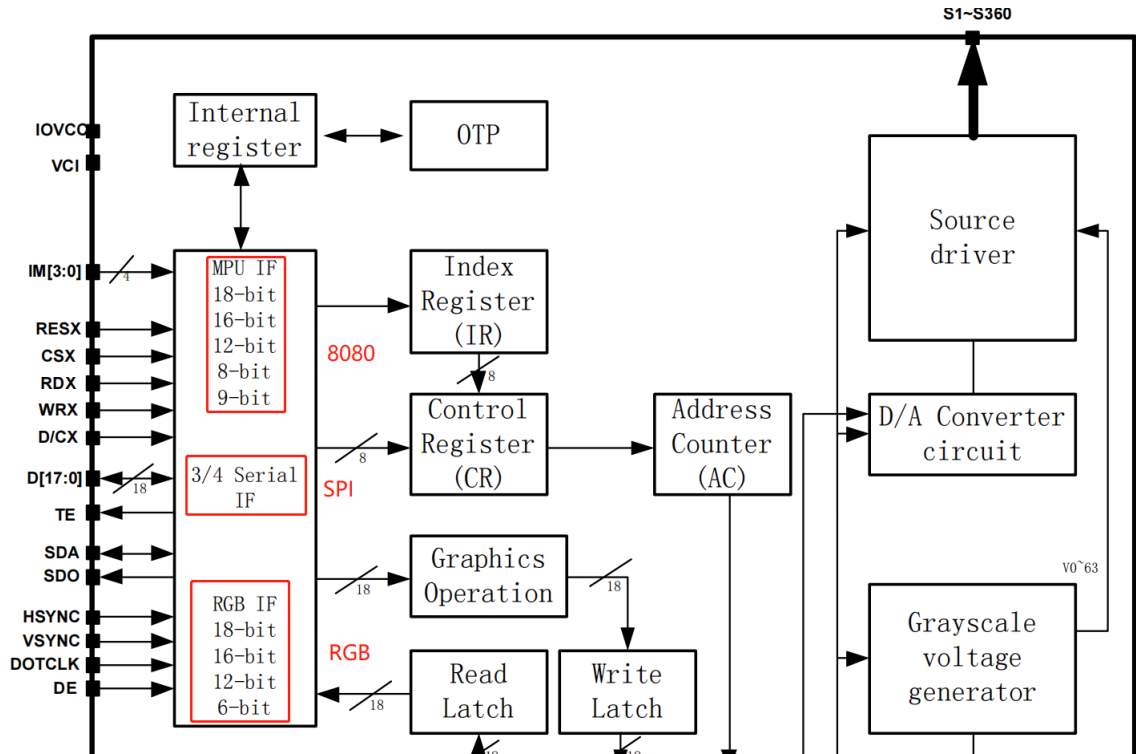


图 7: GC9A01 的硬件框图

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：

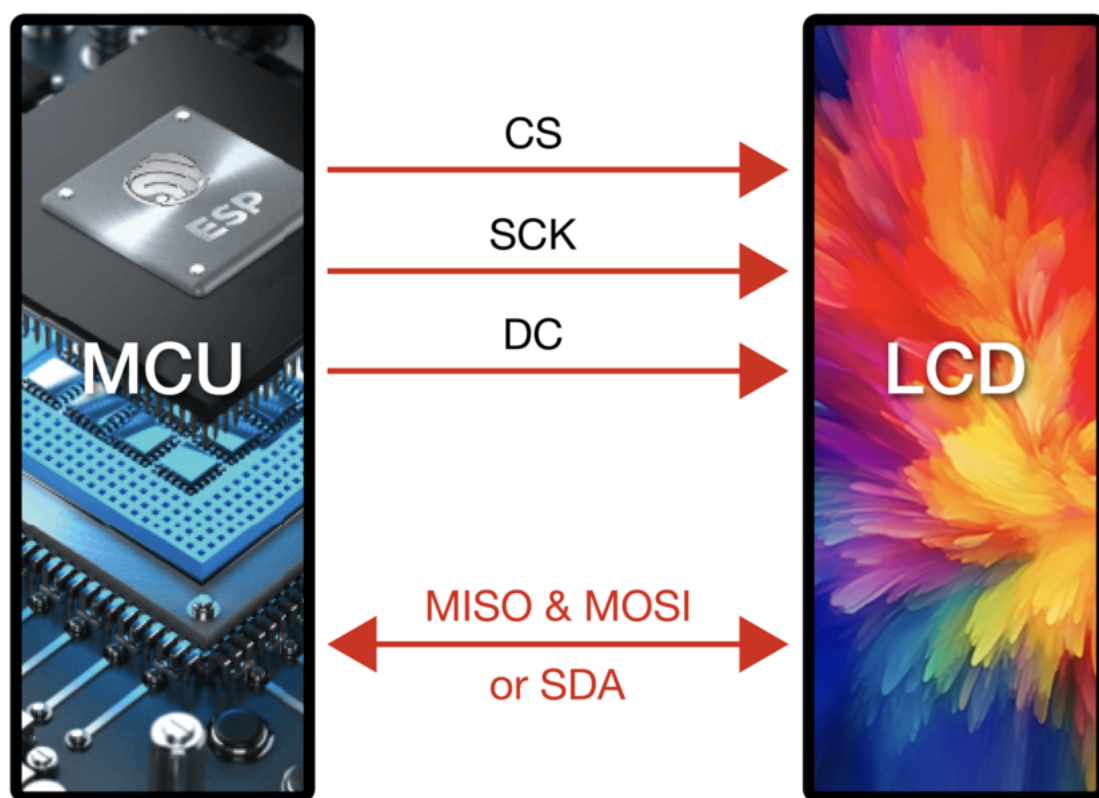


图 9: SPI 接口典型连接示意图

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 MISO。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：

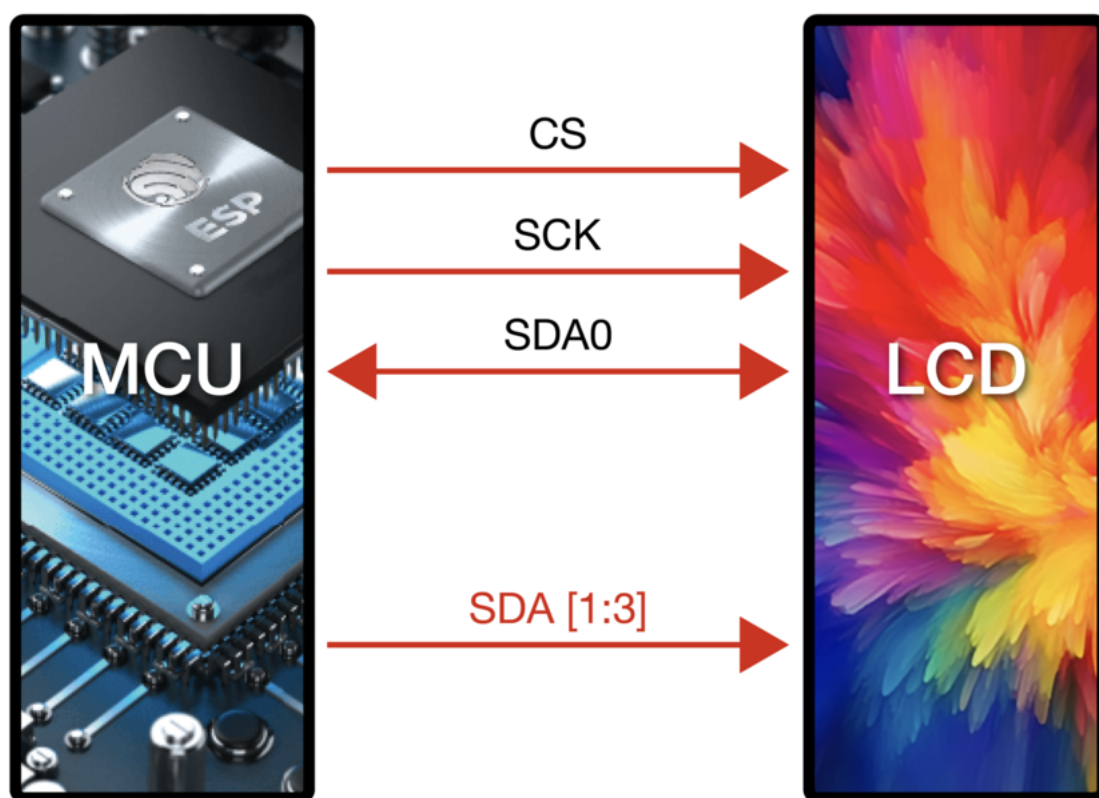


图 10: QSPI 接口典型连接示意图

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：

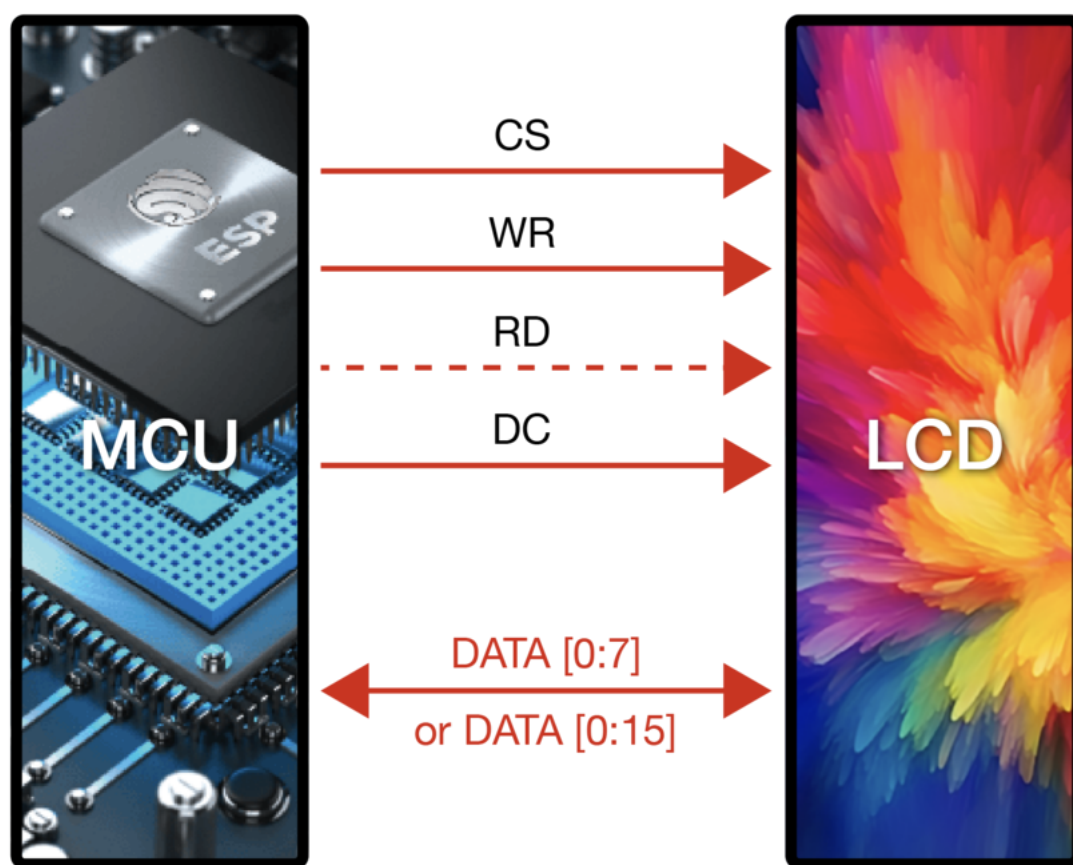


图 11: I80 接口典型连接示意图

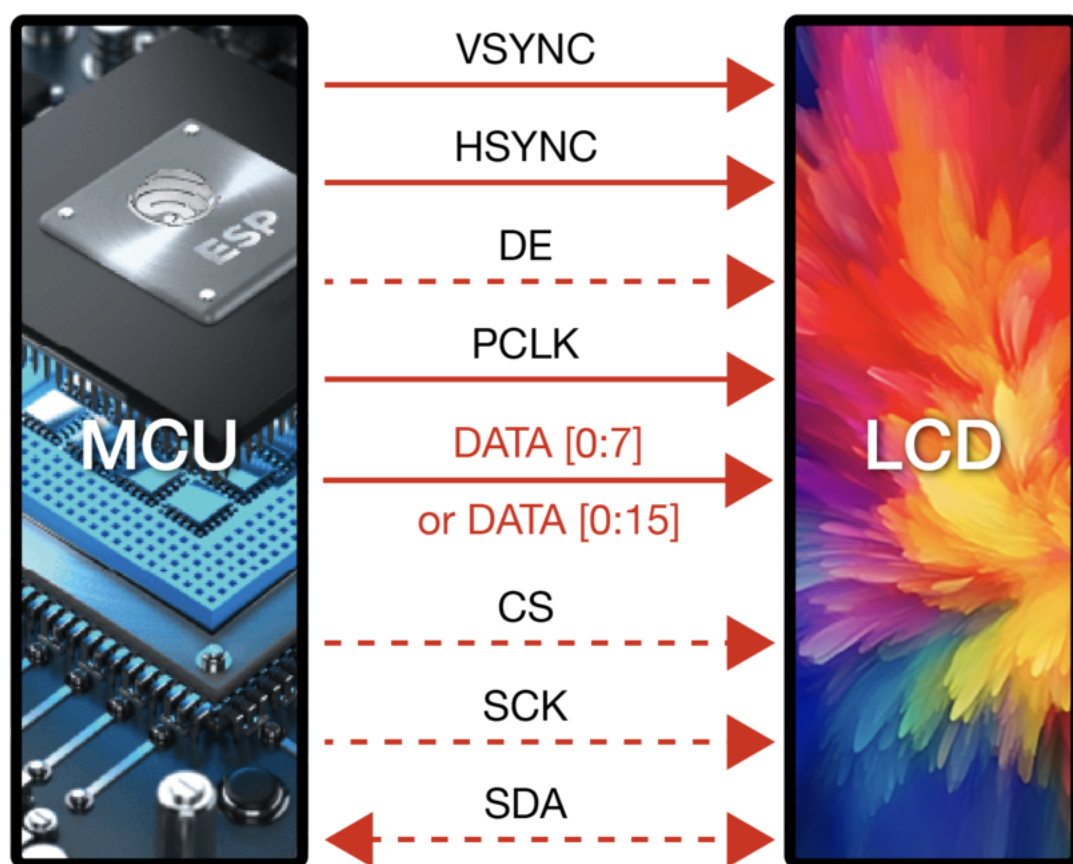


图 12: RGB 接口典型连接示意图

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：

$$\text{接口帧率} = \frac{\text{传输数据量}}{\text{传输时间}}$$

对于 SPI/I80 接口：

$$\text{接口帧率} = \frac{\text{帧大小} \times \text{帧率}}{\text{帧大小} \times \text{帧率} \times \text{帧大小}}$$

对于 RGB 接口：

$$\begin{aligned} \text{接口帧率} &= \frac{\text{帧大小} \times \text{帧率}}{\text{帧大小} \times \text{帧率} \times \text{帧大小}} \\ \text{帧大小} &= \text{帧大小} + \text{帧大小} + \text{帧大小} + \text{帧大小} \\ \text{帧大小} &= \text{帧大小} + \text{帧大小} + \text{帧大小} + \text{帧大小} \end{aligned}$$

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。

备注:

- 若需要在无 LCD 的情况下开发，可以通过 [esp_lcd_usb_display](#) 组件通过 USB UVC 在 PC 显示器上模拟 LCD 的显示效果，从而调试应用程序。其对应示例为 [usb_lcd_display](#)

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
GRAM	Graphic RAM 的缩写，用于保存图形数据的存储区域
TE	用于指示屏幕刷新垂直同步或水平同步的时序信号
Porch	在显示一行或一帧图像数据之前的空白时间间隔
VSYNC	垂直同步信号
HSYNC	水平同步信号

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 , gc9d01 , ili9341 , ssd1681 , st7796 , gc9107
QSPI	axs15231b , gc9b71 , sh8601 , spd2010 , st77903 , st77916 , st77922 , co5300
I80	axs15231b , st7789 , nt35510 , ra8875 , st7796
MIPI-DSI	ek79007 , jd9165 , jd9365 , st7701 , st7703 , st77922 , ili9881c , hx8399 , co5300
3-wire SPI + RGB	st7701 , st77903_rgb , st77922 , gc9503 , nv3052

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：

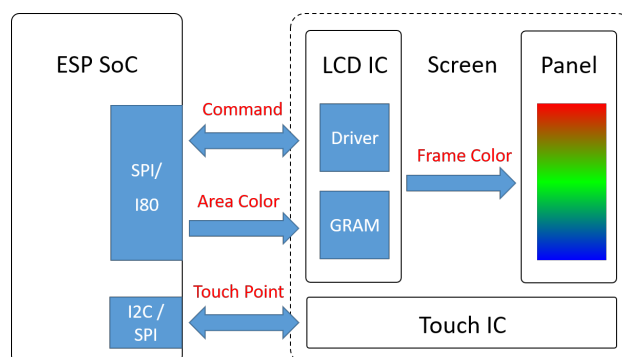


图 13: 硬件驱动框架示意图 - SPI/I80 LCD

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：

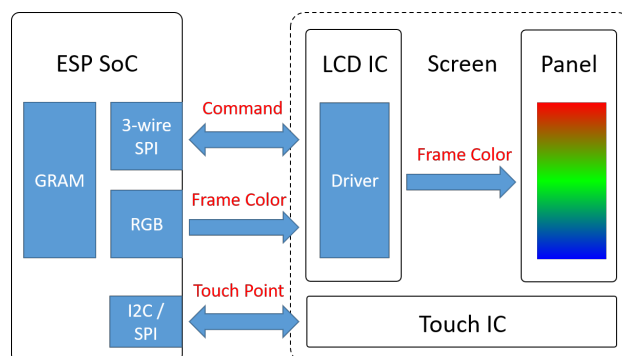


图 14: 硬件驱动框架示意图 - RGB LCD

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：

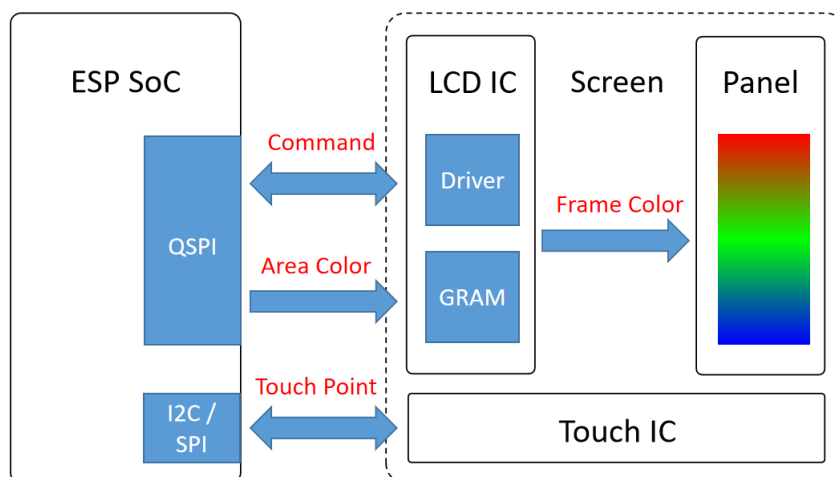


图 15: 硬件驱动框架示意图 - QSPI LCD (有 GRAM)

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。

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。

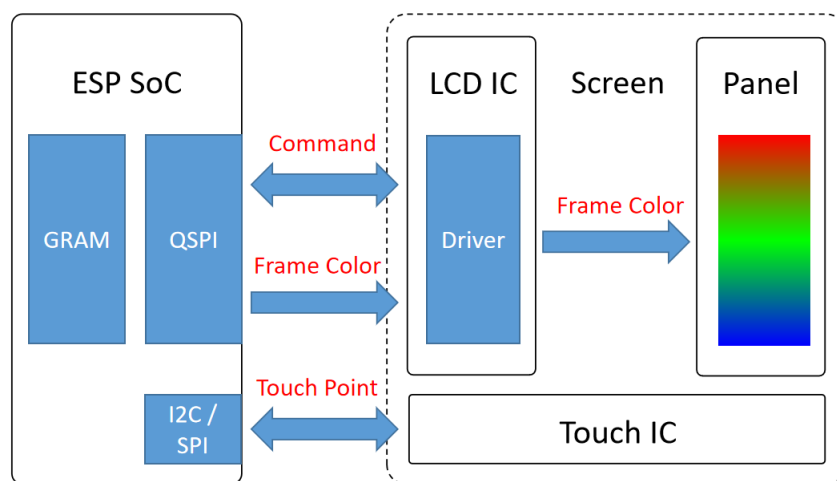


图 16: 硬件驱动框架示意图 - QSPI LCD (无 GRAM)

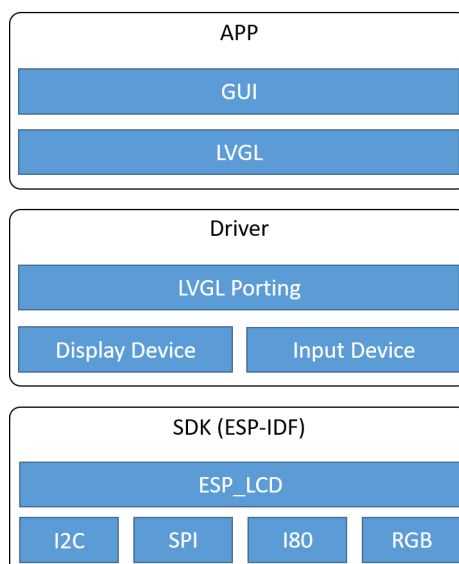


图 17: 软件开发框架示意图

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的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-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

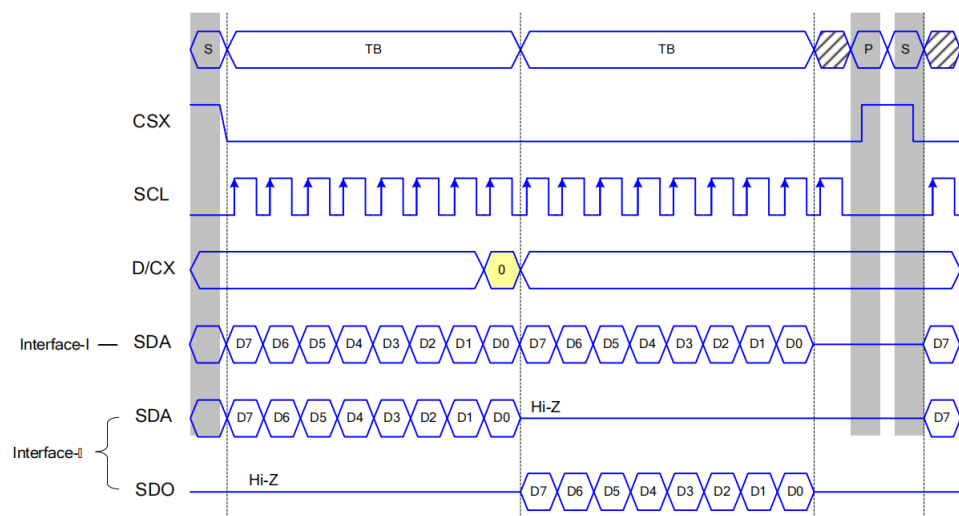


图 20: Interface I/II 模式的时序图对比（4-line）

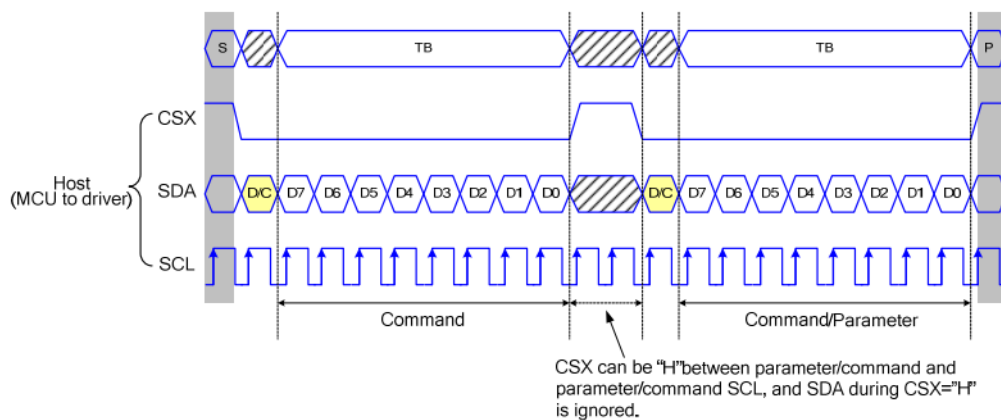


Figure 13 3-line serial interface write protocol (write to register with control bit in transmission)

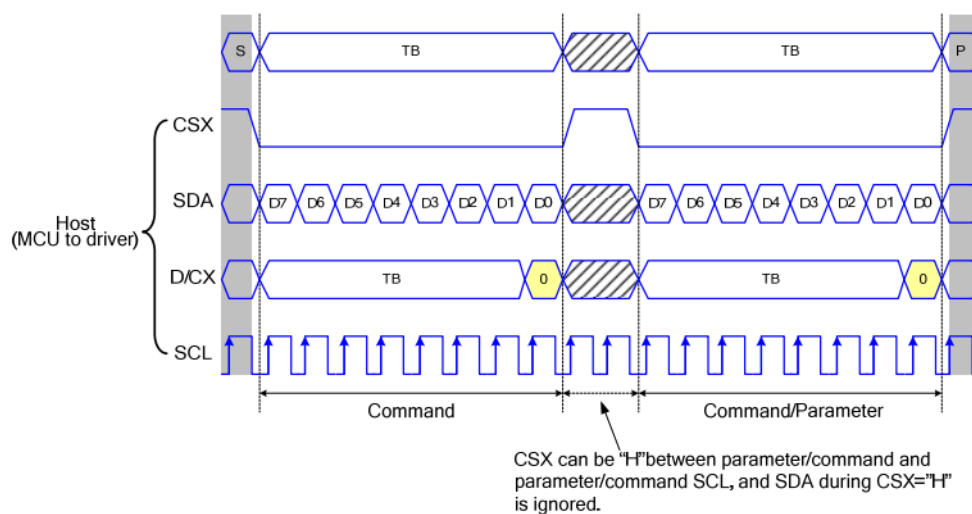


Figure 14 4-line serial interface write protocol (write to register with control bit in transmission)

图 21: 3/4-line 模式的时序图对比 (Interface I)

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。

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)
                                                    // 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
<code>reset()</code>	<code>esp_lcd_panel_reset()</code>	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 <code>LCD_CMD_SWRESET (01h)</code> 进行软件复位。
<code>init()</code>	<code>esp_lcd_panel_init()</code>	通过发送一系列的命令及参数来初始化 LCD 设备。
<code>del()</code>	<code>esp_lcd_panel_del()</code>	释放驱动占用的资源，包括申请的存储空间和使用的 IO。
<code>draw_bitmap()</code>	<code>esp_lcd_panel_draw_bitmap()</code>	首先通过命令 <code>LCD_CMD_CASET (2Ah)</code> 和 <code>LCD_CMD_RASET (2Bh)</code> 发送图像的起始和终止坐标，然后通过命令 <code>LCD_CMD_RAMWR (2Ch)</code> 发送图像数据。
<code>mirror()</code>	<code>esp_lcd_panel_mirror()</code>	通过命令 <code>LCD_CMD_MADCTL (36h)</code> 设置是否镜像屏幕的 X 轴和 Y 轴。
<code>swap_xy()</code>	<code>esp_lcd_panel_swap_xy()</code>	通过命令 <code>LCD_CMD_MADCTL (36h)</code> 设置是否交换屏幕的 X 轴和 Y 轴。
<code>set_gap()</code>	<code>esp_lcd_panel_set_gap()</code>	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。
<code>invert_color()</code>	<code>esp_lcd_panel_invert_color()</code>	通过命令 <code>LCD_CMD_INVON (21h)</code> 和 <code>LCD_CMD_INVOFF (20h)</code> 实现像素的颜色数据按位取反 (<code>0xF0F0 -> 0x0F0F</code>)。
<code>disp_on_off()</code>	<code>esp_lcd_panel_disp_on_off()</code>	通过命令 <code>LCD_CMD_DISON (29h)</code> 和 <code>LCD_CMD_DISOFF (28h)</code> 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}
```

(下页继续)

```

//      {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
    .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_ <i>SPI</i> (fall)	D[0~23]
	0	1	0	RGB+9b_ <i>SPI</i> (fall)	D[0~23]
	0	1	1	RGB+16b_ <i>SPI</i> (rise)	D[0~23]
	1	0	1	MIPI	HSSI_D1_P/N,HSSI_D0_P/N
	1	1	0	MIPI+16b_ <i>SPI</i> (rise)	HSSI_D1_P/N,HSSI_D0_P/N
1	0	0	1	RGB+8b_ <i>SPI</i> (rise)	D[0~23]
	0	1	0	RGB+9b_ <i>SPI</i> (rise)	D[0~23]
	0	1	1	RGB+16b_ <i>SPI</i> (fall)	D[0~23]
	1	0	1	MIPI	HSSI_D1_P/N,HSSI_D0_P/N
	1	1	0	MIPI+16b_ <i>SPI</i> (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）。

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

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。

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

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

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

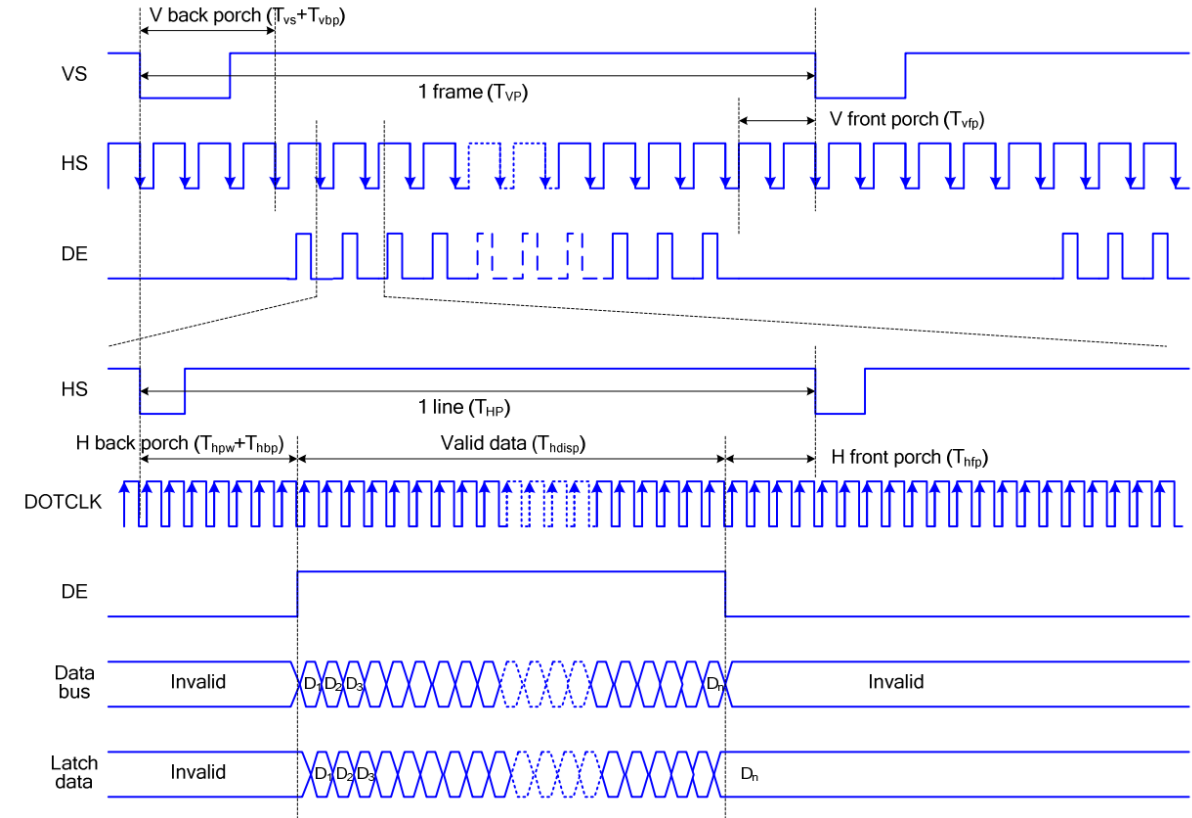
色彩格式

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：

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：

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)**。



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

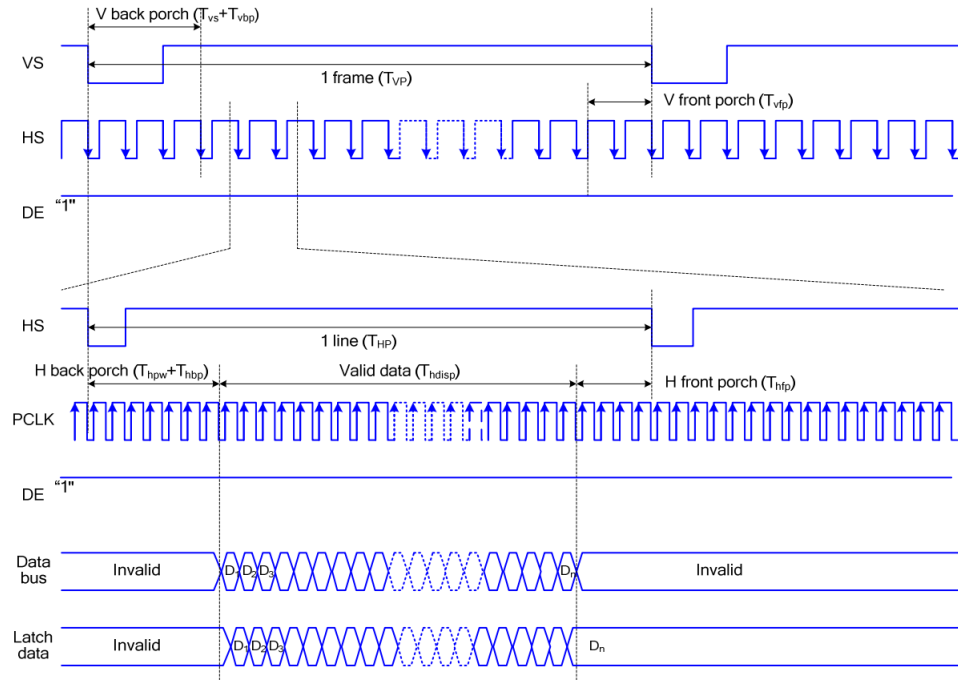


Figure 24 Timing chart of RGB interface HV mod

图 27: ST7701S SYNC 模式的时序图

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

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

serial RGB 565					
Pin	1 st Data	2 nd Data	3 rd Data	...	(3N+1) th 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 st Data	2 nd Data	3 rd Data	...	(3N+1) th 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 st Data	2 nd Data	3 rd Data	...	(3N+1) th 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

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
        // .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));
```

备注：针对 ESP32-P4 RGB 屏幕使用场景的说明：

- 由于 ESP32-P4 有多路 VDDPST，其控制对应区域的 GPIO 最大输出电平（对应关系见 Datasheet）。因此如果不配置 VDDPST，可能部分 GPIO 最大输出 1.8V 导致引脚电压不足，从而导致 RGB 屏幕颜色异常。
- ESP32-S3 最大可支持 30 M 的 RGB 时钟频率。而 ESP32-P4 的 RGB 时钟频率无 30M 限制，因此接口帧率可以更高，可支持更大分辨率的 RGB 屏幕。

相关文档

- [ST7701S 数据手册](#)
- [ST77903 数据手册](#)
- [GC9503 数据手册](#)

4.1.6 MIPI DSI LCD 详解

目录

- 术语表
 - [MIPI-DSI 相关术语](#)
- [MIPI-DSI 接口与协议介绍](#)
 - [MIPI-DSI 接口简介](#)
 - [MIPI-DSI 屏幕刷新原理](#)
 - [MIPI-DSI 协议](#)

- 色彩格式
 - 说明
 - 常见色彩格式与数据类型
- *MIPI-DSI* 驱动流程
 - *DSI*、*DBI* 与 *DPI*
 - 初始化 *DSI* 总线
 - 配置 *DBI* 接口
 - 移植驱动组件
 - 初始化 *LCD* 设备
- 常见问题
 - 问题：屏幕无图像显示，同时无其他报错日志怎么办
 - 问题：屏幕初始化命令无法正常全部发出
 - 问题：屏幕画面扭曲错位
 - 问题：屏幕闪蓝屏，并出现日志：“*can't fetch data from external memory fast enough, underrun happens*”
 - 问题：屏幕画面撕裂如何解决
 - 问题：帧率优化配置
 - 问题：如何驱动默认 *4 lane* 的屏幕
 - 问题：如何确认屏幕所支持的 *lane bit rate* 范围
- 相关文档以及示例

术语表

请参阅 *LCD* 术语表。

MIPI-DSI 相关术语

缩写	全称	说明
MIPI-DSI	MIPI Display Serial Interface	MIPI 联盟开发的高速串行接口标准，专门用于连接处理器和显示设备
DCS	Display Command Set	MIPI-DSI 定义的一套标准化显示指令集，用于控制显示模块
DBI	Display Bus Interface	将 DSI 的 Command Mode 封装成 esp-lcd 的控制 IO 层
DPI	Display Pixel Interface	将 DSI 的 Video Mode 封装成 esp-lcd 的数据面板层
HS	High-Speed Mode	高速模式，主要用于传输显示帧数据，速率一般为 80 Mbps ~ 1.5 Gbps/Lane
LP	Low-Power Mode	低功耗模式，通常用于发送 DCS 命令或控制链路状态，速率最大约 10 MHz
HS-TX/HS-RX	High-Speed Transmitter/Receiver	高速发送/接收器
LP-TX/LP-RX	Low-Power Transmitter/Receiver	低功耗发送/接收器
LP-CD	Low-Power Contention Detector	低功耗竞争检测器
BTA	Bus Turnaround	总线翻转，用于双向通信
ULPS	Ultra Low Power State	超低功耗状态，用于最大限度节省能耗
LPDT	Low Power Data Transmission	低功耗数据传输模式
ESC	Escape Mode	转义模式，MIPI-DSI 的低功耗操作模式
SoT	Start of Transmission	传输开始信号
EoT	End of Transmission	传输结束信号
DI	Data Identifier	数据标识，用于标识数据包类型
ECC	Error Correction Code	错误检测码，用于数据完整性校验
DT	Data Type	数据类型，用于标识像素数据格式
HSYNC	Horizontal Synchronization	水平同步信号
VSYNC	Vertical Synchronization	垂直同步信号
DMA	Direct Memory Access	直接内存访问，用于高效数据传输
PSRAM	Pseudo Static RAM	伪静态随机存取存储器
RGB565	Red Green Blue 565	16 位色彩格式，5 位红色，6 位绿色，5 位蓝色
RGB666	Red Green Blue 666	18 位色彩格式，红绿蓝各 6 位
RGB888	Red Green Blue 888	24 位色彩格式，红绿蓝各 8 位

MIPI-DSI 接口与协议介绍

MIPI-DSI 接口简介 MIPI-DSI (MIPI Display Serial Interface) 是由 MIPI 联盟开发的高速串行接口标准，专门用于连接处理器和显示设备。它具有以下特点：

- 高速传输：每个数据通道可支持高达数 Gbps 的传输速率
- 低功耗：采用低压差分信号，在空闲或低速传输时进入低功耗模式
- 引脚精简：相比并行 RGB 接口，大大减少了信号线数量
- 可扩展性：支持 1 至 4 个数据通道
- 高兼容性：支持多种分辨率和显示设备

目前在乐鑫芯片中，ESP32-P4 拥有 MIPI-DSI 接口，支持 2-lane × 1.5Gbps，并使用 Video 模式输出视频流。

MIPI-DSI 屏幕刷新原理 MIPI-DSI 是一种常用的显示接口标准，用于连接处理器和显示屏。它支持两种主要的数据传输模式：

1. Command 模式 (Command Mode)
2. Video 模式 (Video Mode)

其中 ESP32-P4 目前仅支持使用 Video 模式输出视频流，

在 Video 模式（Video Mode）中，主机以连续视频帧的形式传输数据，与显示屏的刷新周期（如 VSync）严格同步。显示屏无需显存，硬件设计简单。该模式被广泛用于高分辨率、高刷新率的显示场景。

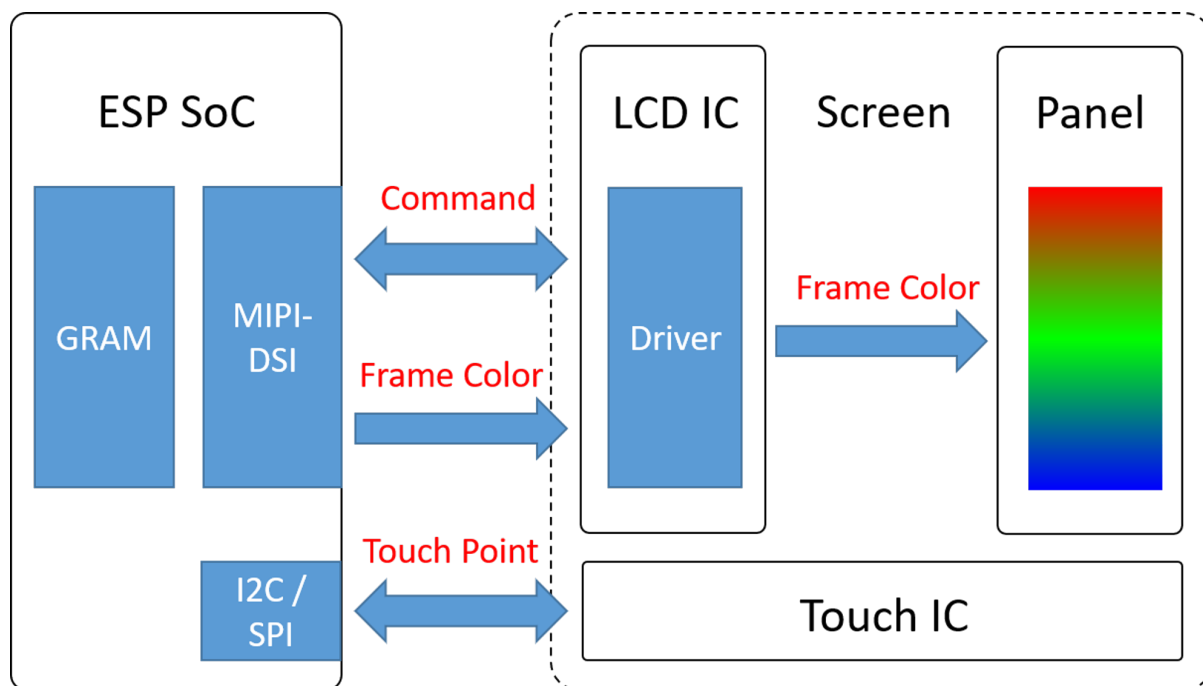


图 31: MIPI-DSI 屏幕刷新的简要流程

上图展示了 MIPI-DSI 屏幕刷新的简要流程，其中：

- 命令数据：通过 DBI 传输 MIPI-DCS 命令到 LCD 控制器，用于初始化寄存器和控制显示功能。
- 帧颜色数据：在 MIPI-DSI 视频模式下，像素数据由主控侧帧缓冲（内/外部 RAM）通过 DMA 连续输出到 LCD 面板进行显示，LCD 驱动 IC 则无需 GRAM。
- 触摸数据：触控信号由触摸屏检测后，通过 I²C 或 SPI 接口传回主控系统进行处理。

MIPI-DSI 协议 MIPI-DSI 可大致分为物理层（Physical Layer）、协议层（Protocol Layer）和显示命令层（DCS Layer），每层都有特定功能，协同完成从主控到显示屏的命令与像素数据传输。

各层的协作流程可概括为下图：

物理层（Physical Layer）

- 功能：负责实现数据的实际传输，是协议栈的底层。采用 MIPI D-PHY 提供高速差分信号传输，支持多通道数据传输。
- 传输模式：
 - 高速模式（High-Speed Mode）：主要用于传输显示帧数据，也可传输命令，速率一般为 80 Mbps ~ 1.5 Gbps/Lane。
 - 低功耗模式（Low-Power Mode）：通常用于发送 DCS 命令（命令模式下）或控制链路状态，速率最大可达约 10 MHz（具体取决于芯片支持）。
- 配置：
 - 时钟通道（Clock Lane）：提供同步信号。
 - 数据通道（Data Lane）：传输实际显示数据。
- 关于通道模块（Lane Module），即 D-PHY 的详细说明
 - 类型

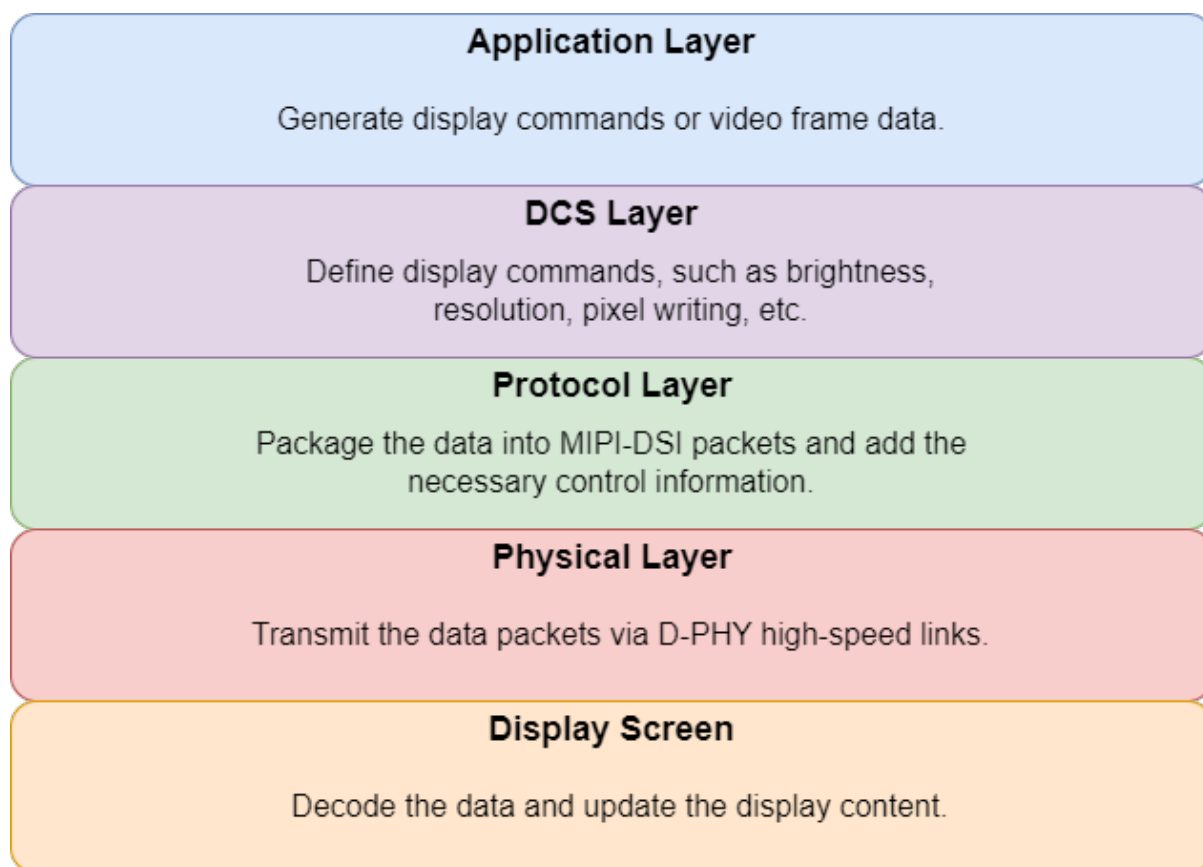


图 32: MIPI-DSI 协议分层

lane 类型/传输角 色	Master	Slave
单向时钟 Lane	HS-TX, LP-TX	HS-RX, LP-RX
单向数据 Lane	HS-TX, LP-TX	HS-RX, LP-RX
双向数据 Lane	HS-TX, LP-TX, HS-RX, LP-RX, LP-CD	HS-TX, LP-TX, HS-RX, LP-RX, LP-CD

备注：各模块功能说明：

- **LP-TX/LP-RX**：低功耗发送/接收器
- **HS-TX/HS-RX**：高速发送/接收器
- **LP-CD**：低功耗竞争检测器

时钟 Lane 通常是单向的，由 Master 提供，Slave 接收。双向 Lane 和 LP-CD 主要用于命令模式下的低功耗通信。

- 状态
 - LP 状态 (LP-00, LP-01, LP-10, LP-11)：使用单端信号，主要用于低速命令传输、控制信号、初始化或待机状态。不同的 LP 状态用于编码和链路控制。
 - HS 状态 (HS-0, HS-1)：使用差分信号，主要用于高速传输视频数据或帧内容。HS-0 和 HS-1 对应差分对上的逻辑电平，仅在 HS-TX/HS-RX 上有效。
- 典型电压
 - LP：典型电压约 0 – 1.2 V
 - HP：典型差分电压约 100 – 300 mV
- 操作模式
 - Escape Mode：只能在 Low-Power Mode，MIPI-DSI 的低功耗操作模式，用于特殊数据传输或触

发特定功能。可支持以下功能：

功能名称	描述	对应命令
LPDT（低功耗数据传输）	在低功耗模式下传输数据	0x87
ULPS（超低功耗状态）	进入超低功耗状态以最大限度节省能耗	0x84
Trigger Command	触发特定事件，例如状态切换或模式调整	0x85

* 进入过程（信号状态机过渡示意）：LP-11 → LP-10 → LP-00 → LP-01 → LP-00

* 退出过程：LP-10 → LP-11

– High-Speed (Burst) Mode：始终在 High-Speed Mode，用于高速串行数据传输，适合大带宽需求。

* 进入过程：LP-11 → LP-01 → LP-00 → SoT (Start of Transmission) → HSD (80Mbps ~ 1.5Gbps)

* 退出过程：EoT (End of Transmission) → LP-11

– Control Mode：默认在 Low-Power Mode，用于停止状态下的命令传输和双向通信（BTA，Bus Turnaround）。

* BTA 传输过程：LP-11 → LP-10 → LP-00 → LP-10 → LP-00（总线翻转状态过渡）

* BTA 接收过程：LP-00 → LP-10 → LP-11

重要信号说明

在 MIPI-DSI 通信过程中，两种关键的 Stop 信号确保了总线的正确状态管理：

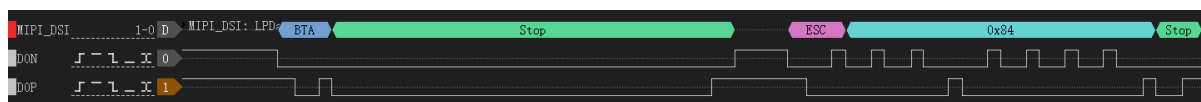


图 33: MIPI-DSI Stop 信号时序图

1. BTA-Stop 信号

当从机完成 Bus Turnaround 响应后发送此信号，表示：

- 总线翻转操作已成功完成
- 从机已释放总线控制权
- 主机可以重新获得总线控制

2. ESC-0X84-Stop 信号

用于指示 Lane 停止信号传输，具体功能包括：

- 使系统进入超低功耗状态（ULPS）
- 对应 Escape Mode 中的 0x84 命令
- 实现最大程度的节能管理

小技巧： 这些 Stop 信号是 MIPI-DSI 协议中确保通信可靠性和功耗管理的关键机制。

协议层 (Protocol Layer)

- 功能：实现数据打包和解析，定义传输格式（如 DCS 和视频数据包），并提供错误检测机制以确保数据的正确性和可靠性。

• 数据帧结构

– 短包 (Short Packet)：

* 长度：固定长度为 4 字节

* 组成：数据标识 (DI)，1 字节；帧数据，2 字节；错误检测 (ECC)，1 字节。

* 示例：

该短帧示例的关键字段如下（按出现顺序）：

* ESC：进入 Escape mode

* 0x87：LPDT（低功耗数据传输模式）

* 0x37：数据标识，对应 DCS 命令“Set Maximum Return Packet Size”，功能：设置主机从从机接收数据时返回数据包的最大大小

* 0x03, 0x00：帧数据，表示设置返回数据包的最大大小为 3 字节

* 0x01：ECC（错误检测码）

– 长包 (Long Packet)：

* 长度：长度可变，6 至 65541 字节。

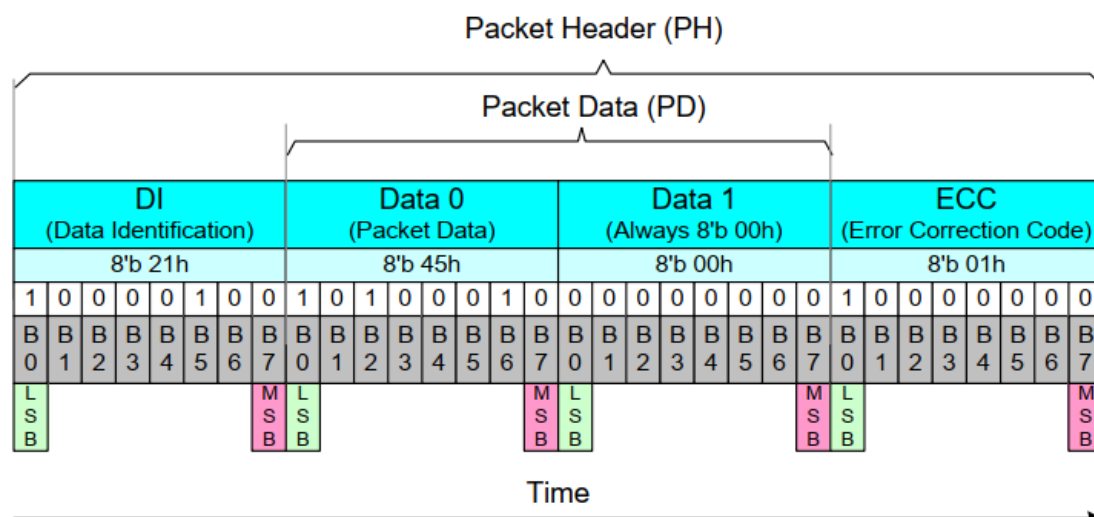


图 34: MIPI-DSI 短帧组成



图 35: MIPI-DSI 短帧时序

- * 组成：帧头（数据标识（DI）1 字节、数据计数（WC）2 字节、错误检测（ECC）1 字节）；数据填充（0 至 65535 字节）；帧尾（校验和（Checksum）2 字节）。
- * 示例：
该长帧示例的关键字段如下（按出现顺序）：
 - * ESC：进入 Escape mode
 - * 0x87：LPDT（低功耗数据传输模式）
 - * 0x39：数据标识，对应 DCS 命令“DCS Write Long”
 - * 0x03, 0x00：数据计数，表示要发送 3 字节的数据命令
 - * 0x09：ECC（错误检测码）
 - * 0xB6, 0xB2, 0xB2：实际发送的数据命令
 - * 0xEF, 0xFA：校验和字段
- 常见的数据标识（DI）

数据标识 (Data ID)	名称	说明
0x05	DCS Short Write, 0 Param	发送一个 DCS 命令, 无参数 (例如 Sleep Out 0x11)
0x15	DCS Short Write, 1 Param	发送一个 DCS 命令, 带 1 个参数
0x39	DCS Long Write	发送一个 DCS 命令, 附带多个字节数据 (常用于内存写入)
0x06	DCS Short Read, 0 Param	读取一个 DCS 状态或寄存器, 无参数
0x16	DCS Short Read, 1 Param	读取一个 DCS 状态或寄存器, 带 1 个参数
0x37	DCS Read Response	显示器返回的 DCS 读数据
0x03	Generic Short Write, 0 Param	发送通用命令, 无参数
0x13	Generic Short Write, 1 Param	发送通用命令, 带 1 个参数
0x23	Generic Short Write, 2 Param	发送通用命令, 带 2 个参数
0x29	Generic Long Write	发送通用命令, 附带多个字节数据
0x04	Generic Read, 0 Param	通用读命令, 无参数
0x14	Generic Read, 1 Param	通用读命令, 带 1 个参数
0x24	Generic Read, 2 Param	通用读命令, 带 2 个参数

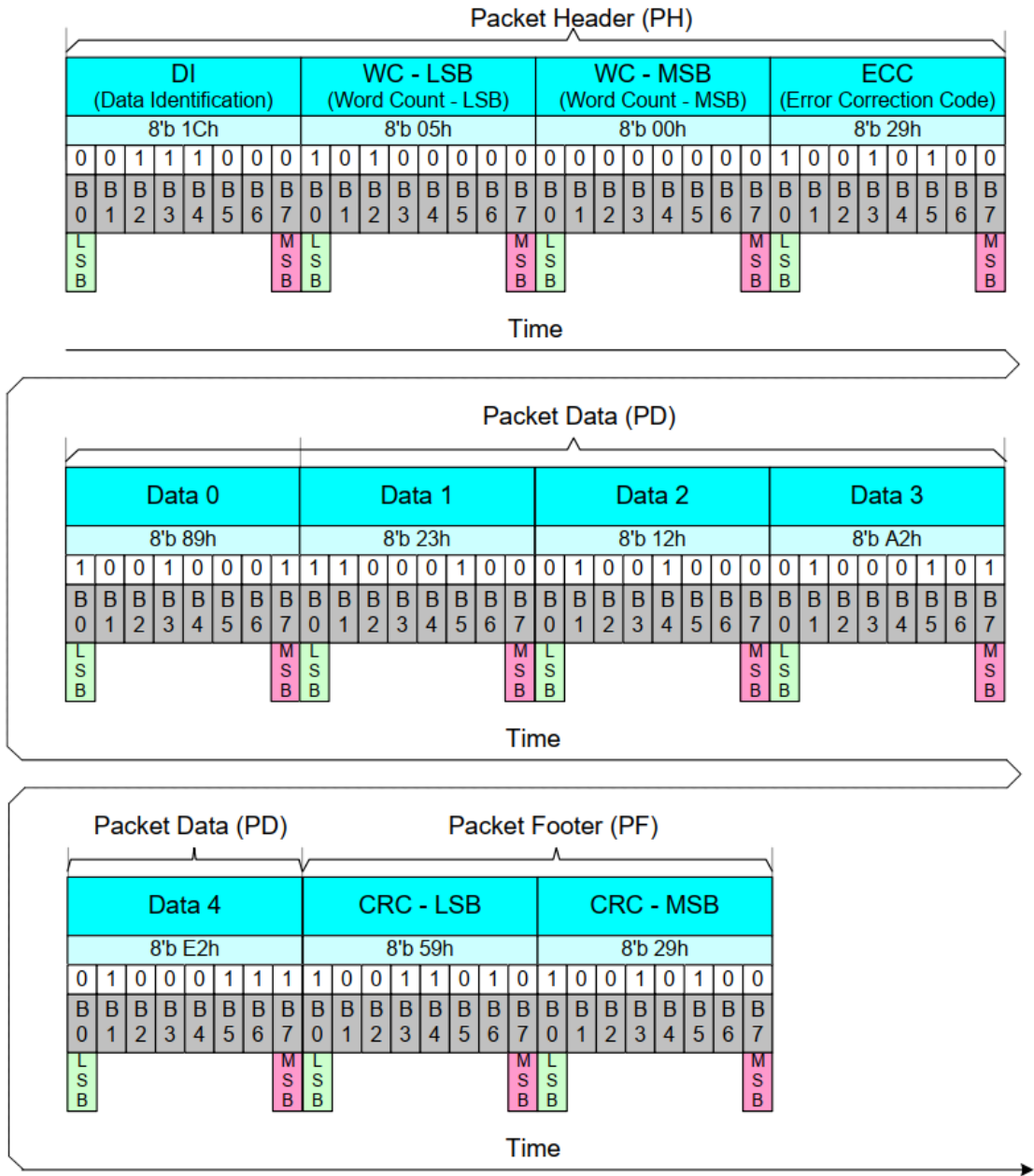


图 36: MIPI-DSI 长帧组成

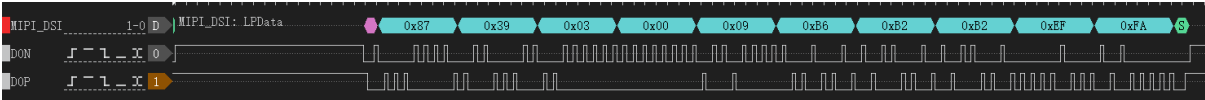


图 37: MIPI-DSI 长帧时序

显示命令层 (DCS Layer)

- 功能：DCS (Display Command Set) 是 MIPI-DSI 定义的一套标准化显示指令集，主控通过这些命令与显示屏交互，用于实现显示模块的基本控制（如开关机、休眠、背光、显示开关）、显示参数配置（如像素格式、显示模式、地址设置），以及显存访问（如写入图像数据、读取状态信息）。DCS 命令通过短写、长写和读命令的形式在 DSI 总线上传输，是应用层与显示硬件之间的主要控制接口。
- 常见命令：

命令码	名称	功能说明
0x01	Software Reset	软件复位
0x11	Sleep Out	退出休眠模式
0x28	Display Off	关闭显示
0x29	Display On	打开显示
0x2A	Column Address Set	设置列地址范围
0x2B	Page Address Set	设置页地址范围
0x2C	Memory Write	写入显存数据
0x2E	Memory Read	读取显存数据
0x36	Address Mode	设置内存扫描方向与翻转方式
0x3A	Pixel Format Set	设置像素格式（如 RGB565/888）

色彩格式

	Display image data (24 bits)																							
	R7	R6	R5	R4	R3	R2	R1	R0	G7	G6	G5	G4	G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0
24-bit	R[7]	R[6]	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	G[7]	G[6]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[7]	B[6]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
18-bit EPF[1:0]=00 (Note 1)	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	0	0	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	0	0	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]	0	0
18-bit EPF[1:0]=01 (Note 2)	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	1	1	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	1	1	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]	1	1
18-bit EPF[1:0]=10	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	R[5]	R[4]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	G[5]	G[4]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]	B[5]	B[4]
16-bit EPF[1:0]=00 (Note 3)	R[4]	R[3]	R[2]	R[1]	R[0]	0	0	0	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	0	0	B[4]	B[3]	B[2]	B[1]	B[0]	0	0	0
16-bit EPF[1:0]=01 (Note 4)	R[4]	R[3]	R[2]	R[1]	R[0]	1	1	1	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	1	1	B[4]	B[3]	B[2]	B[1]	B[0]	1	1	1
16-bit EPF[1:0]=10	R[4]	R[3]	R[2]	R[1]	R[0]	R[4]	R[3]	R[2]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	G[5]	G[4]	B[4]	B[3]	B[2]	B[1]	B[0]	B[4]	B[3]	B[2]

图 38: MIPI-DSI 色彩格式

大多数 MIPI-DSI LCD 支持多种输入色彩格式，如 RGB565、RGB666、RGB888。主控可通过 DCS 命令 0x3A (Pixel Format Set) 配置显示的颜色深度，例如：

- 设置为 **RGB565 (16-bit)** 时，可发送 Command: 0x3A, Parameter: 0x55（具体参数值需参考 LCD 控制器规格书）。

像素数据通常通过 **长数据包 (Long Packet)** 传输，数据包中包含像素数据和对应的数据类型 (Data Type)。例如，RGB565 格式的像素流一般使用 **DT = 0x0E (Packed Pixel Stream, 16-bit)**。

说明

- **16-bit** 与 **18-bit** 格式常用于降低带宽和存储占用
- **24-bit** 格式适用于高质量显示场景

常见色彩格式与数据类型

色彩格式	位深	DCS 参数 (例)	Data Type (DT)
RGB565	16-bit	0x55	0x0E
RGB666	18-bit	0x66	0x1E / 0x2E
RGB888	24-bit	0x77	0x3E

MIPI-DSI 驱动流程

MIPI-DSI LCD 驱动流程可大致分为四个部分：初始化 DSI 总线、配置 DBI 接口、移植驱动组件和初始化 LCD 设备。

DSI、DBI 与 DPI

- DSI：负责 PHY、Host、Bridge 的底层初始化，它是 DBI 与 DPI 的公共“母线”。
- DBI：将 DSI 的 Command Mode 封装成 esp-lcd 的控制 IO 层
- DPI：将 DSI 的 Video Mode 封装成 esp-lcd 的数据面板层，负责帧缓冲到屏的持续刷新

初始化 DSI 总线

```
esp_lcd_dsi_bus_handle_t dsi_bus = NULL;
esp_lcd_dsi_bus_config_t bus_config = {
    .bus_id = 0,                                // DSI 控制器编号
    .num_data_lanes = 2,                        // 数据通道数量 (1-2)
    .phy_clk_src = MIPI_DSI_PHY_CLK_SRC_DEFAULT, // PHY 时钟源
    .lane_bit_rate_mbps = 1300,                // 每通道比特率 (Mbps)
};
ESP_ERROR_CHECK(esp_lcd_new_dsi_bus(&bus_config, &dsi_bus));
```

其中数据通道数量 (num_data_lanes) 值取决于 LCD IC 所使用并支持的通道数，部分 LCD IC 可通过寄存器配置所使用的通道数，而一些 LCD IC 所使用的通道数是固定的。值得注意的是，ESP32-P4 最大支持 2 lanes，即支持 1-lane 和 2-lane 两种 LCD 通道配置。

每通道比特率 (lane_bit_rate_mbps) 计算方法：

- 示例计算：
 - 分辨率: 800×1280 (hspw=4, hbp=20, hfp=20; vspw=4, vbp=20, vfp=20)
 - 刷新率: 60Hz
 - 色深: 24bpp
 - 数据通道数: 2
- 计算原始数据速率
 - pixel_clock = 水平总像素 × 垂直总像素 × 刷新率
 - raw_data_rate = pixel_clock × bits_per_pixel
 示例计算：
 - * pixel_clock = (800+20+20+4) × (1280+20+20+4) × 60 = 62.4MHz
 - * raw_data_rate = 62.4M × 24 = 1497.6Mbps
- 考虑 DSI 协议开销

- 8b/10b 编码: 实际数据率需要乘以 1.25
- 协议包头和 ECC: 额外增加约 10-15% 开销
- $\text{protocol_data_rate} = \text{raw_data_rate} \times 1.25 \times 1.15$

示例计算:

* $\text{protocol_data_rate} = 1497.6 \times 1.25 \times 1.15 = 2153.55\text{Mbps}$

- 分配到各数据通道

- $\text{lane_bit_rate} = \text{protocol_data_rate} \div \text{num_data_lanes}$

示例计算:

* $\text{lane_bit_rate} = 2153.55 \div 2 = 1076.78\text{Mbps}$

- 预留带宽余量 (建议 20%)

- $\text{final_bit_rate} = \text{lane_bit_rate} \times 1.2$

示例计算:

* $\text{final_bit_rate} = 1076.78 \times 1.2 \approx 1292\text{Mbps}$

因此建议 `lane_bit_rate_mbps` 值取 `final_bit_rate`, 在示例计算场景下为 1300 左右。注意, 在 ESP32P4 平台下, `lane_bit_rate_mbps` 最大值不能超过 1500, 同时推荐最小值不要低于 480。

配置 DBI 接口

```
esp_lcd_panel_io_handle_t io_handle = NULL;
esp_lcd_dbi_io_config_t io_config = {
    .virtual_channel = 0,           // 虚拟通道号 (0-3), 大多数 LCD 仅支持 `0`
    ↪ 通道, 通常设为 `0` 即可
    .lcd_cmd_bits = 8,             // 命令位宽
    .lcd_param_bits = 8,          // 参数位宽
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_dbi(dsi_bus, &io_config, &io_handle));
```

其中命令和参数位宽必须与面板规格匹配, 虚拟通道用于区分多个设备。

移植驱动组件 移植 MIPI-DSI 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_dbi_io_config_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:

对于大多数 MIPI-DSI 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 MIPI-DSI 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 `ek79007` 替换为 `ili9881`

初始化 LCD 设备 下面以 **EK79007** 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_ek79007.h" // 目标驱动组件的头文件
```

(下页继续)

```

// static const ek79007_lcd_init_cmd_t lcd_init_cmds[] = {
// {cmd, { data }, data_size, delay_ms}
// {0xE0, (uint8_t []){0x00}, 1, 0},
// {0xE1, (uint8_t []){0x93}, 1, 0},
// {0xE2, (uint8_t []){0x65}, 1, 0},
// {0xE3, (uint8_t []){0xF8}, 1, 0},
// ...
// };

ESP_LOGI(TAG, "MIPI DSI PHY Powered on");
esp_ldo_channel_handle_t ldo_mipi_phy = NULL;
esp_ldo_channel_config_t ldo_mipi_phy_config = {
    .chan_id = 3,
    .voltage_mv = 2500,
};
ESP_ERROR_CHECK(esp_ldo_acquire_channel(&ldo_mipi_phy_config, &ldo_mipi_phy));

ESP_LOGI(TAG, "Initialize MIPI DSI bus");
esp_lcd_dsi_bus_handle_t mipi_dsi_bus = NULL;
esp_lcd_dsi_bus_config_t bus_config = EK79007_PANEL_BUS_DSI_2CH_CONFIG();
ESP_ERROR_CHECK(esp_lcd_new_dsi_bus(&bus_config, &mipi_dsi_bus));

ESP_LOGI(TAG, "Install panel IO");
esp_lcd_panel_io_handle_t mipi_dbi_io = NULL;
esp_lcd_dpi_panel_config_t dpi_config = {
    .virtual_channel = 0,
    .dpi_clk_src = MIPI_DSI_DPI_CLK_SRC_DEFAULT,
    .dpi_clock_freq_mhz = 60, // 像素时钟频率
    .in_color_format = LCD_COLOR_FMT_RGB888, // 色彩格式
    .video_timing = {
        .h_size = 800, // 水平分辨率
        .v_size = 1280, // 垂直分辨率
        .hsync_pulse_width = 4, // HSYNC 脉冲宽度
        .hsync_back_porch = 20, // 水平后肩
        .hsync_front_porch = 20, // 水平前肩
        .vsync_pulse_width = 4, // VSYNC 脉冲宽度
        .vsync_back_porch = 20, // 垂直后肩
        .vsync_front_porch = 20, // 垂直前肩
    },
    .flags.use_dma2d = true, // 使用 2D DMA 加速
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_dbi(mipi_dsi_bus, &dbi_config, &mipi_dbi_
↪io));

ESP_LOGI(TAG, "Install EK79007S panel driver");
esp_lcd_panel_handle_t panel_handle = NULL;
ek79007_vendor_config_t vendor_config = {
    .flags = {
        .use_mipi_interface = 1,
    },
    .mipi_config = {
        .dsi_bus = mipi_dsi_bus,
        .dpi_config = &dpi_config,
    },
};
const esp_lcd_panel_dev_config_t panel_config = {
    .reset_gpio_num = EXAMPLE_LCD_IO_RST, // Set to -1 if not use
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB, // Implemented by LCD_
↪command `36h`
    .bits_per_pixel = EXAMPLE_LCD_BIT_PER_PIXEL, // Implemented by LCD_
↪command `3Ah` (16/18/24)

```

(下页继续)

(续上页)

```

        .vendor_config = &vendor_config,
    };
    ESP_ERROR_CHECK(esp_lcd_new_panel_ek79007(mipi_dbi_io, &panel_config, &panel_
↪handle));
    ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
    ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
    ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));

```

注意，在 ESP32P4 中要求为 MIPI DSI PHY 提供 2.5V 的稳定电源。其中时钟频率计算参考上文 `pixel_clock` 计算方式。色彩格式、分辨率、脉冲宽度和前后肩参数需要严格遵循面板规格书要求。

然后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 MIPI-DSI 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 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 MIPI-DSI 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 MIPI-DSI 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 MIPI-DSI 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，[MIPI-DSI 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_dpi_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_dpi_panel_set_pattern()`：设置预定义的图案到屏幕上，用于测试或调试目的。
- `esp_lcd_dpi_panel_set_color_conversion()`：为 DPI 面板设置颜色转换配置。
- `esp_lcd_dpi_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_color_trans_event(esp_lcd_panel_handle_t panel, const esp_
↪lcd_dpi_panel_event_callbacks_t *edata, void *user_ctx)
{
    /* 可以在此处进行一些操作 */

    return false;
}

static bool example_on_refresh_event(esp_lcd_panel_handle_t panel, const esp_lcd_
↪dpi_panel_event_callbacks_t *edata, void *user_ctx)
{
    /* 可以在此处进行一些操作 */

    return false;
}

esp_lcd_dpi_panel_event_callbacks_t cbs = {
    .on_color_trans_done = example_on_color_trans_event,    //↪
    ↪当用户的颜色缓冲区被复制到内部帧缓冲区时触发
    .on_refresh_done = example_on_refresh_event,            //↪
    ↪当内部帧缓冲区的内容完成刷新并显示到屏幕上时触发
};
ESP_ERROR_CHECK(esp_lcd_dpi_panel_register_event_callbacks(panel_handle, &cbs, &
↪example_user_ctx));

```

常见问题

问题：屏幕无图像显示，同时无其他报错日志怎么办 需要检查 SPIRAM SPEED 值是否配置到 200M。驱动 MIPI-DSI 接口屏幕需要 PSRAM 有更高的带宽。

问题：屏幕初始化命令无法正常全部发出 芯片 DSI 的数据引脚 (DSI_DATAN0 和 DSI_DATAP0) 和时钟引脚 (DSI_CLKN 和 DSI_CLKP) 与屏幕的数据引脚和时钟引脚没有正确连接。

问题：屏幕画面扭曲错位 这种情况一般是 lane_bit_rate_mbps 与 pixel_clock 不匹配，或 lane_bit_rate_mbps 与屏幕要求速率不匹配造成的，可调整 lane_bit_rate_mbps 数值并测试，具体可参考上文 pixel_clock 计算方式。

问题：屏幕闪蓝屏，并出现日志：“can't fetch data from external memory fast enough, underrun happens”

该问题出现的原因是 PSRAM 带宽不足，可以考虑降低 lane_bit_rate_mbps 和 pixel_clock 数值，如果在 RGB888 颜色格式场景下，可以考虑更改为 RGB565 格式。同时考虑配置以下选项以改善 PSRAM 带宽：

- CONFIG_SPIRAM_XIP_FROM_PSRAM=y
- CONFIG_CACHE_L2_CACHE_256KB=y
- CONFIG_CACHE_L2_CACHE_LINE_128B=y
- COMPILER_OPTIMIZATION_PERF=y

问题：屏幕画面撕裂如何解决 参考 LCD 屏幕撕裂详解，相关例程可参考 [mipi_dsi_avoid_tearing](#)。

问题：帧率优化配置 可考虑配置以下选项以优化帧率：

- CONFIG_SPIRAM_XIP_FROM_PSRAM=y
- CONFIG_CACHE_L2_CACHE_256KB=y
- CONFIG_CACHE_L2_CACHE_LINE_128B=y
- COMPILER_OPTIMIZATION_PERF=y
- LVGL 相关配置：
 - CONFIG_LV_MEMCPY_MEMSET_STD=y
 - CONFIG_LV_MEM_CUSTOM=y

问题：如何驱动默认 4 lane 的屏幕 大多数 LCD IC 都支持通过寄存器调整 lane 数，但是不同 IC 的寄存器地址与操作并不相同，需要根据具体 IC 的 datasheet 或者直接咨询屏厂确认能否或者是否可调整 lane 数。例如 EK79007 中，可通过配置 0xB2 寄存器选择使用 2 lane/4 lane。

RB2h: Panel Control Register									
Address (MIPI I/F)	B2h					Access Attribute			R/W
						Number of Parameter(s)			1
Parameter	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]	Default Value
	—	NBW (0)	—	2Lane_EN (0)	—	—	—	—	00h
Description	NBW: Normally black or normally white setting. NBW="1" : Normally black. NBW="0" : Normally white(default). 2Lane_EN: 2 lane/4 lane setting 2Lane_EN="0":4 lane(default) 2Lane_EN="1":2 lane								

图 39: 通道数配置

问题：如何确认屏幕所支持的 lane bit rate 范围 在高速串行通信里，UI 通常指信号比特（bit）在物理通道上的时间单位，也就是一个 bit 时间（bit period）。例如如果帧率或链路速率是 1 Gbps，那么一个 UI ≈ 1 纳秒（ns）。如果速率是 500 Mbps，那么 UI ≈ 2 ns，以此类推。因此可根据 LCD IC datasheet 中 AC ELECTRICAL CHARACTERISTIC - High speed transmission 来推算 lane bit rate 范围，如图所示，其 lane bit rate (单个 Data Lane 的传输速率) 范围为 20Mbps 至 500Mbps。

15.3.3. High speed transmission					
Parameter	Symbol	Min	Typ	Max	Units
UI instantaneous	UI _{INST}	2	-	12.5	ns
Data to Clock Skew(measured at transmitter)	T _{SKEW(TX)}	-0.15	-	0.15	UI _{INST}
Data to Clock Setup time(measured at receiver)	T _{SETUP(RX)}	0.15	-	-	UI _{INST}
Data to Clock Hold time(measured at receiver)	T _{HOLD(RX)}	0.15	-	-	UI _{INST}
20%~80% rise time and fall time	T _R , T _F	150	-	-	ps
		-	-	0.3	UI _{INST}

图 40: 速率范围

相关文档以及示例

- [MIPI DSI 规范](#)
- [ESP-IDF MIPI DSI LCD 编程指南](#)
- [ESP LCD 驱动库](#)
- [ESP LCD 示例代码](#)
- [ESP LCD FAQ](#)

4.1.7 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。

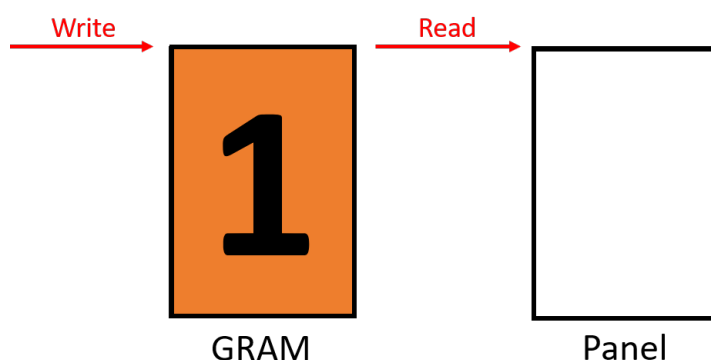


图 41: 简化的屏幕刷新示意图

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	基于多 GRAM 的方法
SPI, I80, QSPI (with internal GRAM)	LCD	基于 TE 信号的方法

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，会向写入发送 TE 信号。当写入接收到 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。

4.1.8 LCD Application Solution

LCD 方案介绍

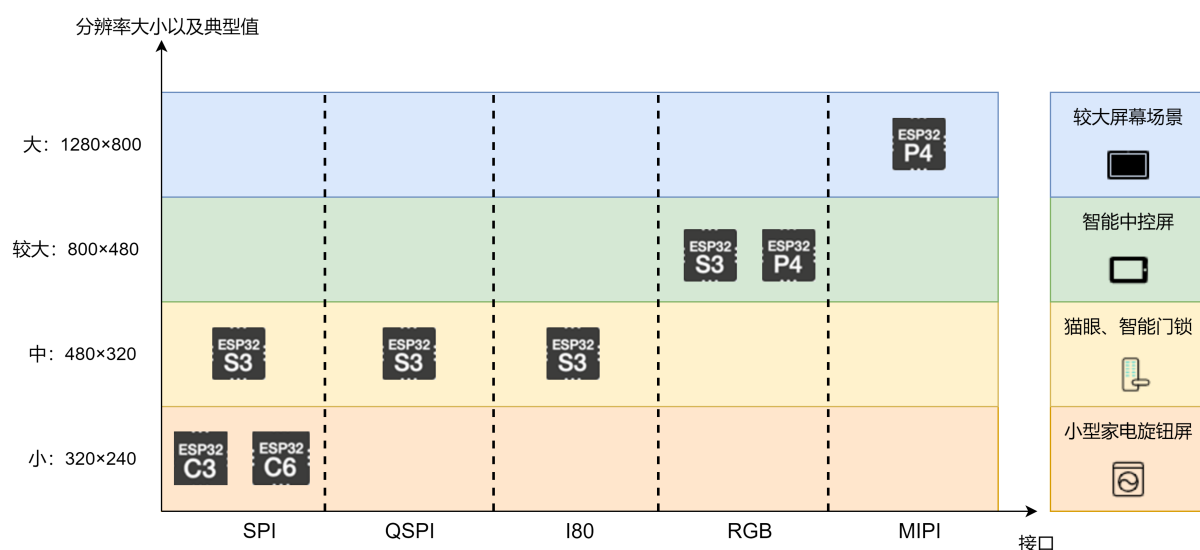
乐鑫 HMI 智能屏 (LCD) 方案具有卓越性能和可扩展性，可与不同 ESP 主控芯片搭配。该方案在智能家居控制、家电屏幕、医疗设备、工业控制和儿童教育等多个应用场景下表现出色。优势包括高性能图形可视化，低内存占用等。此外，屏幕适配方案完善，并支持高性能 JPEG 解码和帧率优化。以下是 LCD 方案具体介绍：

- **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：



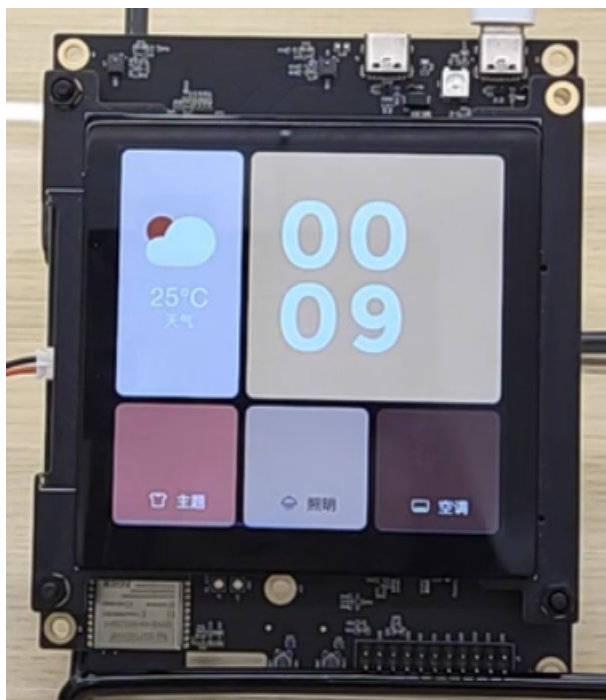
LCD 常见应用场景

乐鑫 LCD 方案广泛应用于各个领域，包括但不限于：

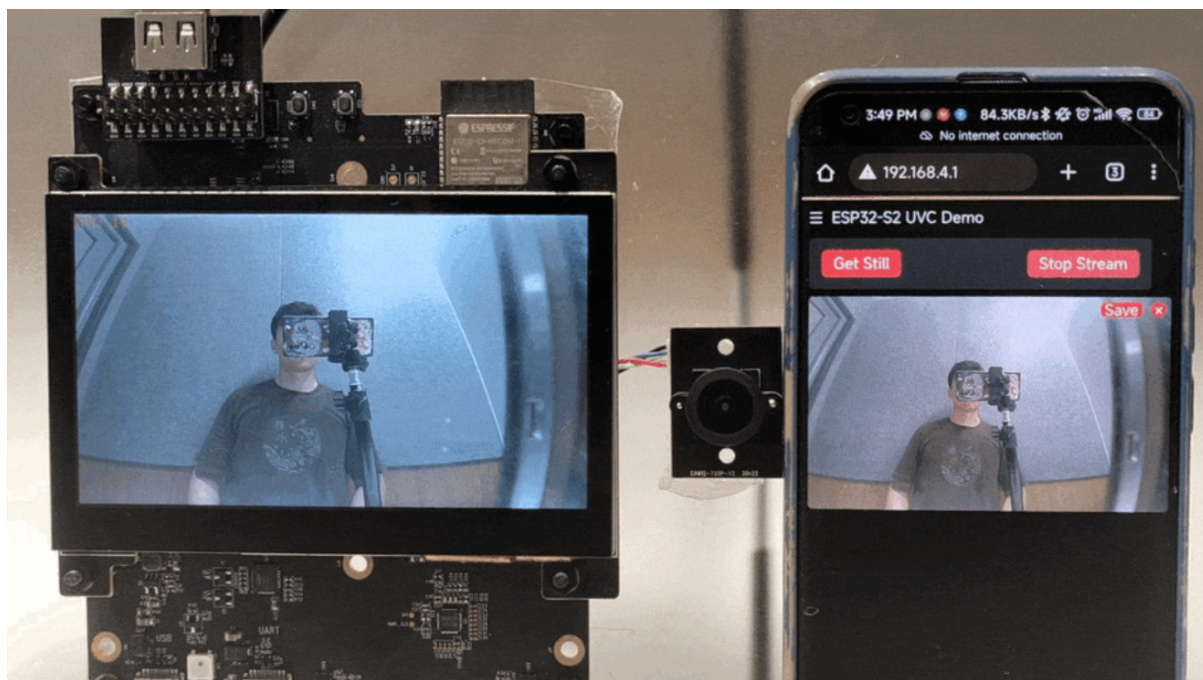
- **旋钮屏方案：**针对智能家电产品，传统段码屏和黑白屏升级首选。支持 Wi-Fi、蓝牙，扩展接口可实现串口通讯等功能。典型应用场景为小型家电应用中的旋钮屏。



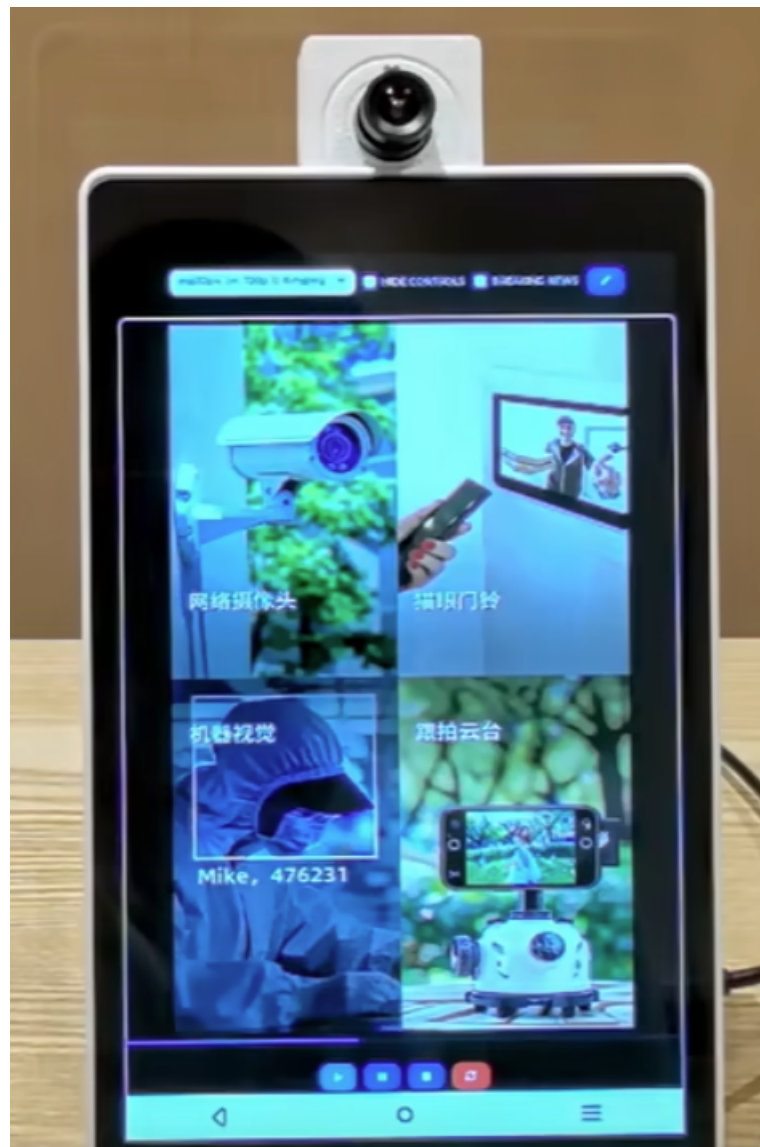
- **中控屏方案：**集成 Wi-Fi、BLE、离线语音、RGB LCD 显示，支持离线命令词和连续语音识别。适用于传统 86 面板的升级迭代，构建了集设备控制、开关面板、温控面板、智能遥控器为一体的智能家居控制中枢。



- **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](#)

4.1.9 GUI 优化解决方案

LVGL 防撕裂与帧率优化

- 支持基于 full_fresh、direct_mode、partial_mode 三种模式的防撕裂机制
- 支持针对 RGB 接口和 MIPI-DSI 接口的全屏幕旋转优化

适用场景：

- 需要兼顾防撕裂和帧率优化
- 需要进行 LVGL 整屏旋转的场景

相关示例：

- https://github.com/espressif/esp-iot-solution/tree/master/examples/display/lcd/mipi_dsi_avoid_tearing
- https://github.com/espressif/esp-iot-solution/tree/master/examples/display/lcd/rgb_avoid_tearing

LVGL 图像解码组件

- 支持 LVGL v8 和 v9 版本
- 支持 PNG、JPG、QOI、PJPNG、SJPG、SPNG 等格式解码，并优化了解码速率
- 针对 ESP32-P4 支持 JPEG 硬件解码
- 针对 ESP32-S3 支持 JPEG 解码加速

对于其中特殊格式的说明：

- PJPNG 格式：基于 PNG 格式转换，支持透明度，支持硬件 JPEG 解码
- SJPG 格式、SPNG 格式：分段解码，适用于 ESP32-C 系列 RAM 较小且没有 PSRAM 的场景
- QOI 格式：对比软件 JPEG，QOI 解码速度更快，但是压缩率更低

组件链接：https://github.com/espressif/esp-iot-solution/tree/master/components/display/tools/esp_lv_decoder

相关示例：https://github.com/espressif/esp-iot-solution/tree/master/examples/hmi/perf_benchmark

基于 MMAP 的文件系统与脚本工具

- 自动打包用户文件
- 脚本自动转换格式，支持 SJPG，SPNG，QOI，PJPNG 格式等
- 通过文件系统以 mmap 的方式读取素材

适用场景：

- 不满足于 SPIFFS 等文件系统的读取速度，可使用 mmap 的方式读取素材。
- 需要 SJPG，SPNG，QOI，PJPNG 等格式的图片素材

组件链接：https://components.espressif.com/components/espressif/esp_mmap_assets

相关示例：https://github.com/espressif/esp-iot-solution/tree/master/examples/hmi/perf_benchmark

Thorvg 组件

支持 Lottie 动画和 TVG 矢量图形解析渲染刷图，noglic 版本支持

组件链接：<https://components.espressif.com/components/espressif/thorvg>

相关示例：

- Lottie：<https://components.espressif.com/components/espressif/thorvg/versions/0.13.8/examples/thorvg-example>

freetype_label 组件

基于 freetype 的用户组件，支持字体字号管理，渲染，着色，支持中英文，支持 swap16

适用场景：

- 支持图片和文件的直接渲染

组件链接：<https://github.com/espressif/idf-extra-components/tree/master/freetype>

相关示例：https://github.com/espressif/esp-iot-solution/tree/master/examples/hmi/lvgl_freetype

tinyGL 3D 图形库

- tinyGL 是一个轻量级的 3D 图形库，支持 OpenGL ES 1.1 的子集，适用于 ESP32-P4 平台。

适用场景：

- 需要实现 3D 图形渲染。

组件链接：<https://github.com/espressif2022/TinyGL>

SDL3 开发库

SDL 是一个跨平台的开发库，旨在通过 OpenGL/Direct3D/Metal/Vulkan 提供对音频、键盘、鼠标、摇杆和图形硬件的低级访问。它被视频播放软件、模拟器以及包括 Valve 获奖游戏在内的许多流行游戏和 Humble Bundle 游戏使用。

适用场景：

- 有模拟器开发、游戏移植和跨平台应用开发需求的用户。

组件链接：<https://components.espressif.com/components/georgik/sdl/versions/3.1.2~9>

相关示例：<https://github.com/georgik/esp32-sdl3-test/tree/main>

Lottie 播放组件

lottie_player 是一款专为 LVGL v8 设计的轻量级高效 Lottie 动画播放器。它可以将基于矢量的动画无缝集成到 LVGL 项目中。通过利用 ThorVG 作为渲染引擎，该模块确保了现代嵌入式 GUI 应用程序的高性能和灵活性。

适用场景：

- 使用 LVGL v8 需要播放 Lottie 动画的用户

组件链接：https://components.espressif.com/components/espressif2022/lottie_player

esp_emote_gfx

esp_emote_gfx 是一款面向资源受限平台的轻量级动画渲染库，支持将 GIF 转换为自定义 EAF 动画格式进行播放，具备解码快、内存占用小、显示灵活的优势，适用于 HMI 场景中的高效动态图形播放需求。

特性：

- 支持文字滚动效果
- 支持同时播放多个动图，帧率独立控制
- 适应多尺寸屏幕

适用场景：

- 各类屏幕上的引导动效、开机动画、表情和文字渲染等
- 资源受限平台（ESP32-C2/C3 等）上的动态 UI 组件展示

组件链接：https://components.espressif.com/components/espressif2022/esp_emote_gfx/

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	20×4/16×8 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。

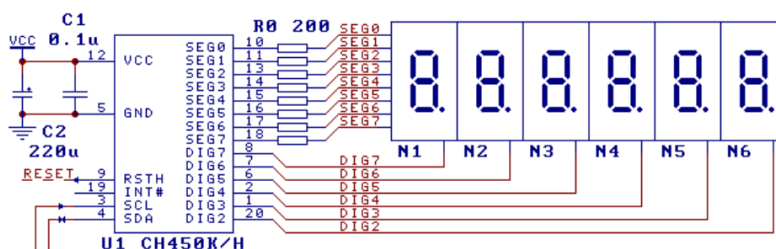


图 42: CH450 典型应用电路图

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 20×4 或 16×8 段码式液晶屏，该芯片通过 I2C 接口与 ESP32 进行通信。

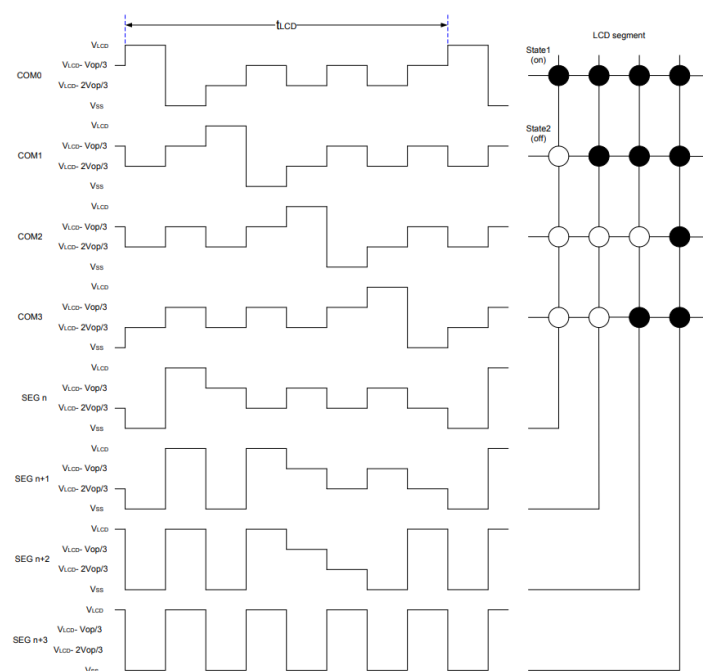


图 43: HT16C21 典型驱动模型

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。

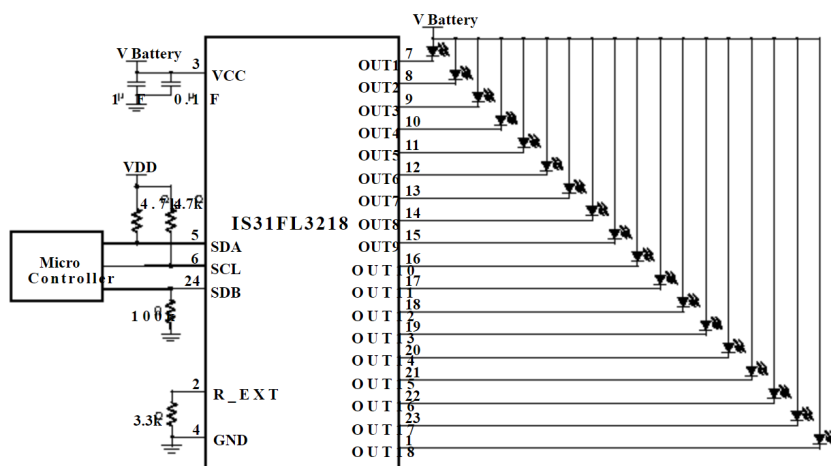


图 44: IS31FL3218 典型应用电路图

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*
 - *Macros*

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
};
```

(下页继续)

(续上页)

```

{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}, // step2: 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。

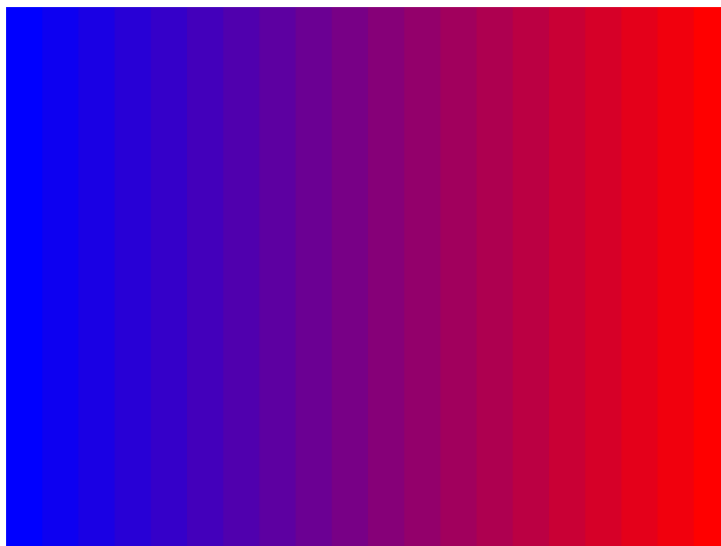
例 1. 定义一个颜色设置，让指示灯显示红色。

```
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
    {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
};
```

(下页继续)

(续上页)

```

    {LED_BLINK_STOP, 0, 0}, // step4: stop blink
    ↪ (blue color)
};

```

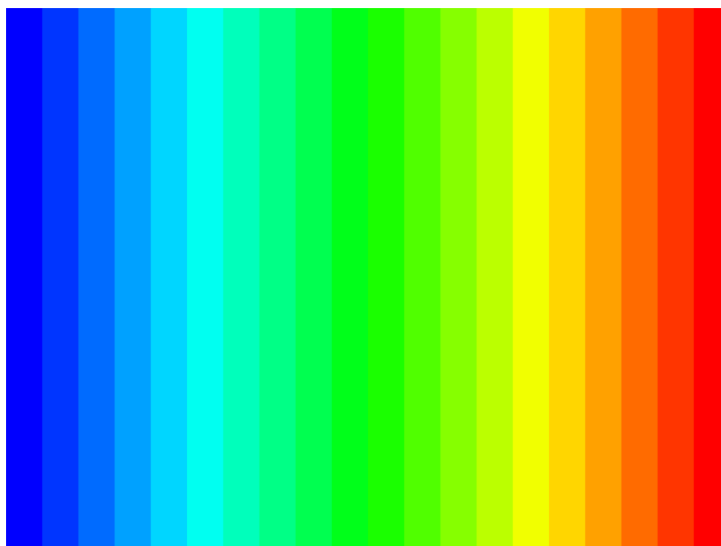
例 4. 定义一个颜色渐变，让指示灯从红色渐变到蓝色，并循环执行。

```

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 `INSERT_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_gpio_config_t led_indicator_gpio_config = {
    .gpio_num = 1,                /**< num of GPIO */
    .is_active_level_high = 1,
};
led_indicator_config_t config = {
    .blink_lists = led_indicator_get_sample_lists(),
    .blink_list_num = led_indicator_get_sample_lists_num(),
};
esp_err_t ret = led_indicator_new_gpio_device(&config, &led_indicator_gpio_config,
↪ &led_handle);

```

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_gpio_config_t led_indicator_gpio_config = {
    .gpio_num = 1,                /**< num of GPIO */
    .is_active_level_high = 1,
};

led_indicator_config_t config = {
    .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](#)

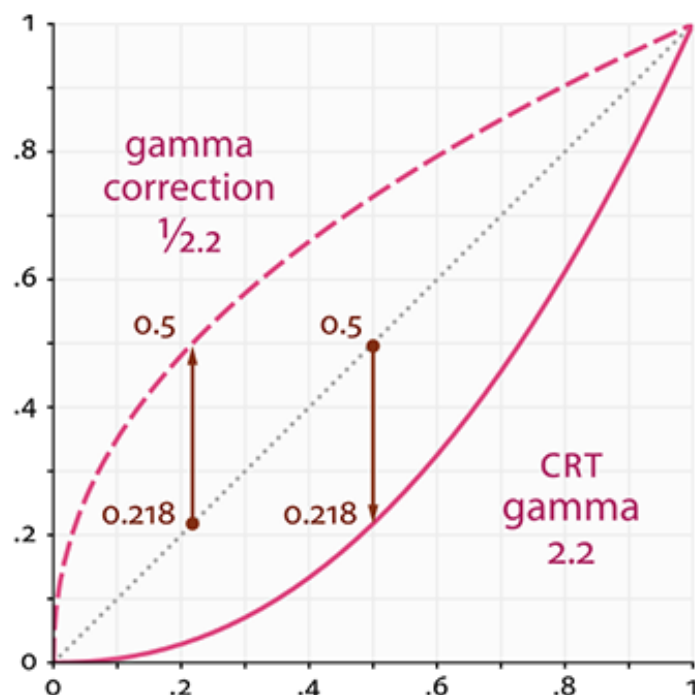


图 45: Gamma 曲线

Functions

`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 ihsv_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.
- **ihsv_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 irgb_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.
- **irgb_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 (0xIIIIIIII) 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

Macros

`LED_INDICATOR_CHECK` (a, str, action)

`LED_INDICATOR_CHECK_WARNING` (a, str, action)

4.4 LCD 工具

4.4.1 ESP LV DECODER

允许使用 LVGL 中的图像。除此之外，它还允许使用自定义格式（称为分割图像），该格式可以在嵌入式系统上以更优化的方式进行解码。

分割图像，参考 [SJPG](#) 的实现。

功能

- 支持标准和自定义分割图像格式，包括 JPG、PNG 和 [QOI](#)。
- 解码标准图像需要与完整未压缩图像大小相当的 RAM，建议用于具有更多 RAM 的设备。
- 分割图像是基于标准图像格式的自定义格式，专为 LVGL 设计。
- 文件存储和 C 数组均实现了文件读取。
- 分割图像分段解码，因此不支持缩放和旋转。

设置图片转换格式

依赖组件 `esp_mmap_assets`。它会在编译过程中自动打包图像并将其转换为所需的格式。

JPG 转换成 SJPG

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".jpg"
    MMAP_SUPPORT_SJPG
    MMAP_SPLIT_HEIGHT 16)
```

PNG 转换成 SPNG

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".png"
    MMAP_SUPPORT_SPNG
    MMAP_SPLIT_HEIGHT 16)
```

PNG、JPG 转换成 QOI

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".jpg, .png"
    MMAP_SUPPORT_QOI)
```

PNG、JPG 转换成 SQOI

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".jpg, .png"
    MMAP_SUPPORT_QOI
    MMAP_SUPPORT_SQOI
    MMAP_SPLIT_HEIGHT 16)
```

应用示例

注册解码器 在 LVGL 启动后注册解码器函数。

```
esp_lv_decoder_handle_t decoder_handle = NULL;
esp_lv_decoder_init(&decoder_handle); //Initialize this after lvgl starts
```

API 参考

Header File

- [components/display/tools/esp_lv_decoder/include/esp_lv_decoder.h](#)

Functions

`esp_err_t esp_lv_decoder_init (esp\_lv\_decoder\_handle\_t *ret_handle)`

Register the 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_decoder_deinit (esp\_lv\_decoder\_handle\_t handle)`

Deinitialize the 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_decoder_handle_t**

Type of handle for the decoder.

4.4.2 ESP LV FS

esp_lv_fs 允许 LVGL 使用文件系统来访问资源。该组件依赖 *esp_mmap_assets*，以高效管理文件操作，并支持多个分区。

功能

- 基于 *esp_mmap_assets* 用于文件系统创建。
- 支持标准文件操作：fopen, fclose, fread, ftell, fseek。
- 使用 *esp_partition_read* API 实现高效文件访问。
- 支持多个分区。

依赖

依赖组件 *esp_mmap_assets*。它提供文件偏移索引关系。

应用示例

注册文件系统 在 LVGL 启动后注册文件系统。

```
#include "esp_lv_fs.h"
#include "esp_mmap_assets.h"

esp_lv_fs_handle_t fs_drive_a_handle;
mmap_assets_handle_t mmap_drive_a_handle;

const mmap_assets_config_t asset_cfg = {
    .partition_label = "assets_A",
    .max_files = MMAP_DRIVE_A_FILES,
    .checksum = MMAP_DRIVE_A_CHECKSUM,
    .flags = {
        .mmap_enable = true,
    }
};

mmap_assets_new(&asset_cfg, &mmap_drive_a_handle);

const fs_cfg_t fs_drive_a_cfg = {
    .fs_letter = 'A',
    .fs_assets = mmap_drive_a_handle,
    .fs_nums = MMAP_DRIVE_A_FILES
};

esp_lv_fs_desc_init(&fs_drive_a_cfg, &fs_drive_a_handle);
```

API 参考

Header File

- [components/display/tools/esp_lv_fs/include/esp_lv_fs.h](#)

Functions

`esp_err_t esp_lv_fs_desc_init` (const *fs_cfg_t* *cfg, *esp_lv_fs_handle_t* *ret_handle)

Initialize file descriptors for the filesystem.

This function sets up the filesystem by initializing file descriptors based on the provided configuration. It allocates necessary memory and populates the file descriptors with information about the assets.

参数

- **cfg** –[in] Pointer to the filesystem configuration structure.
- **ret_handle** –[out] Pointer to the handle that will hold the initialized filesystem instance.

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NO_MEM: Memory allocation failed

`esp_err_t esp_lv_fs_desc_deinit` (*esp_lv_fs_handle_t* handle)

Deinitialize the filesystem and clean up resources.

This function cleans up the filesystem by freeing allocated memory for file descriptors and other resources associated with the provided handle.

参数 **handle** –[in] Handle to the filesystem instance to be deinitialized.

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid argument

Structures

struct **fs_cfg_t**

Configuration structure for the filesystem.

Public Members

char **fs_letter**

Filesystem letter identifier

int **fs_nums**

Number of filesystem instances

mmap_assets_handle_t **fs_assets**

Handle to memory-mapped assets

Type Definitions

typedef void ***esp_lv_fs_handle_t**

Filesystem handle type definition.

4.4.3 ESP MMAP ASSETS

该模块主要用于打包资源（如图像、字体等），将其直接映射以供用户访问。同时，它内部也集成了丰富的图像预处理功能。

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_decoder 来解析。参见组件 [esp_lv_decoder](#)。

设置分片高度

- 当开启 split 图片 (MMAP_SUPPORT_SJPG、MMAP_SUPPORT_SPNG 或 MMAP_SUPPORT_SQOI) 处理功能，可以灵活设置图片分割的高度。

支持 QOI

- 支持将图像从 JPG、PNG 格式转换为 QOI。
- 支持 split qoi，需要使用 esp_lv_decoder 来解析。

支持多分区

- 使用 spiffs_create_partition_assets 挂载分区时，支持挂载多个分区，且为每个分区设置不同的处理逻辑。

支持 LVGL Image Converter

- 通过开启 MMAP_SUPPORT_RAW 以及相关配置，可以将 JPG、PNG 转换为 Bin，以便在 LVGL 中使用它们。

CMake 选项

支持以下选项。这些选项允许您配置图像处理的各个方面。

```
set(options
    FLASH_IN_PROJECT,
    FLASH_APPEND_APP,
    MMAP_SUPPORT_SJPG,
    MMAP_SUPPORT_SPNG,
    MMAP_SUPPORT_QOI,
    MMAP_SUPPORT_SQOI,
    MMAP_SUPPORT_RAW,
    MMAP_RAW_DITHER,
    MMAP_RAW_BGR_MODE)

set(one_value_args
    MMAP_FILE_SUPPORT_FORMAT,
    MMAP_SPLIT_HEIGHT,
    MMAP_RAW_FILE_FORMAT,
    IMPORT_INC_PATH,
    COPY_PREBUILT_BIN)
```

选项说明

通用选项

- FLASH_IN_PROJECT: 用户可以选择在 idf.py flash 时自动将图像与应用程序二进制文件、分区表等一起烧录。
- FLASH_APPEND_APP: 启用将二进制数据 (bin) 附加到应用程序二进制文件 (app_bin)。
- IMPORT_INC_PATH: 指定生成头文件的目标路径。默认与引用组件位置相同。
- COPY_PREBUILT_BIN: 复制预生成的二进制文件到目标目录。此选项允许您使用外部生成的资产二进制文件，而不是从源文件构建。
- MMAP_FILE_SUPPORT_FORMAT: 指定支持的文件格式 (例如，.png、.jpg、.ttf)。
- MMAP_SPLIT_HEIGHT: 定义图像分割的高度以减少内存使用。依赖于：
 - MMAP_SUPPORT_SJPG
 - MMAP_SUPPORT_SPNG
 - MMAP_SUPPORT_SQOI

通用示例

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".jpg,.png,.ttf"
)
```

预构建二进制资源示例

```
spiffs_create_partition_assets(
    my_spiffs_partition
    "${ASSETS_DIR}"
    FLASH_IN_PROJECT
    COPY_PREBUILT_BIN "${ASSETS_DIR}/prebuilt.bin"
)
```

支持的图像格式

- MMAP_SUPPORT_SJPG: 启用对 **SJPG** 格式的支持。
- MMAP_SUPPORT_SPNG: 启用对 **SPNG** 格式的支持。
- MMAP_SUPPORT_QOI: 启用对 **QOI** 格式的支持。
- MMAP_SUPPORT_SQOI: 启用对 **SQOI** 格式的支持。依赖于:
 - MMAP_SUPPORT_QOI

图像分割示例

```
spiffs_create_partition_assets(
    my_spiffs_partition
    my_folder
    FLASH_IN_PROJECT
    MMAP_FILE_SUPPORT_FORMAT ".jpg"
    MMAP_SUPPORT_SJPG
    MMAP_SPLIT_HEIGHT 16
)
```

LVGL Bin 支持

- MMAP_SUPPORT_RAW: 将图像转换为 LVGL 支持的 **Binary** 数据。
 - 参考:**
 - LVGL v8: [详细参考](#)
 - LVGL v9: [详细参考](#)
- MMAP_RAW_FILE_FORMAT: 指定 **RAW** 图像的文件格式。
 - LVGL v8: {true_color, true_color_alpha, true_color_chroma, indexed_1, indexed_2, indexed_4, indexed_8, alpha_1, alpha_2, alpha_4, alpha_8, raw, raw_alpha, raw_chroma}
 - LVGL v9: 未使用。
- MMAP_RAW_COLOR_FORMAT: 指定 **RAW** 图像的颜色格式。
 - LVGL v8: {RGB332, RGB565, RGB565SWAP, RGB888}
 - LVGL v9: {L8, I1, I2, I4, I8, A1, A2, A4, A8, ARGB8888, XRGB8888, RGB565, RGB565A8, ARGB565, RGB888, AUTO, RAW, RAW_ALPHA}
- MMAP_RAW_DITHER: 启用 **RAW** 图像的 **抖动**。
 - LVGL v8: 需要抖动。
 - LVGL v9: 未使用。
- MMAP_RAW_BGR_MODE: 启用 **RAW** 图像的 **BGR 模式**。
 - LVGL v8: 未使用。
 - LVGL v9: 未使用。

LVGL v9 示例

```
spiffs_create_partition_assets(
    .....
    MMAP_FILE_SUPPORT_FORMAT ".png"
    MMAP_SUPPORT_RAW
    MMAP_RAW_COLOR_FORMAT "ARGB8888"
)
```

LVGL v8 示例

```
spiffs_create_partition_assets(
    .....
    MMAP_FILE_SUPPORT_FORMAT ".png"
    MMAP_SUPPORT_RAW
    MMAP_RAW_FILE_FORMAT "true_color_alpha"
    MMAP_RAW_COLOR_FORMAT "RGB565SWAP"
)
```

应用示例

生成头文件 (mmap_generate_my_spiffs_partition.h) 该头文件自动生成，包含内存映射资源的基本定义。

```
#include "mmap_generate_my_spiffs_partition.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 mmap_generate_my_spiffs_partition.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，当然你也可以跳过此检验。

```
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,
    .flags = {
        .mmap_enable = true,
        .app_bin_check = true,
    },
};

ESP_ERROR_CHECK(mmap_assets_new(&config, &asset_handle));
```

资源使用 可以使用 mmap_generate_my_spiffs_partition.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);
```

(下页继续)

(续上页)

```

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.

`size_t mmap_assets_copy_mem(mmap\_assets\_handle\_t handle, size_t offset, void *dest_buffer, size_t size)`

Copy a portion of an asset's memory to a destination buffer.

参数

- **handle** –[in] Asset instance handle.
- **offset** –[in] Offset within the asset's memory from which to start copying.
- **dest_buffer** –[out] Pointer to the destination buffer where the memory will be copied.
- **size** –[in] Number of bytes to copy from the asset's memory to the destination buffer.

返回 The actual number of bytes copied, or 0 if the operation fails.

`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.

int **mmap_assets_get_stored_files** (*mmap_assets_handle_t* handle)

Get the number of stored files in the memory-mapped asset.

This function returns the total number of assets stored in the memory-mapped file associated with the given handle. This can be useful for iterating over all assets or validating the file contents.

参数 **handle** –[in] Asset instance handle. This handle is used to identify the memory-mapped file instance containing the assets.

返回 The number of stored assets. Returns the total count of assets if the handle is valid, otherwise returns 0.

Structures

struct **mmap_assets_config_t**

Asset configuration structure, contains the asset table and other configuration information.

Public Members

const char ***partition_label**

Label of the partition containing the assets

int **max_files**

Maximum number of assets supported

uint32_t **checksum**

Checksum of the asset table for integrity verification

unsigned int **mmap_enable**

Flag to indicate if memory-mapped I/O is enabled

unsigned int **app_bin_check**

Flag to enable app header and bin file consistency check

unsigned int **full_check**

Flag to enable self-consistency check

unsigned int **metadata_check**

Flag to enable metadata verification

unsigned int **reserved**

Reserved for future use

struct *mmap_assets_config_t*::[anonymous] **flags**

Configuration flags

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

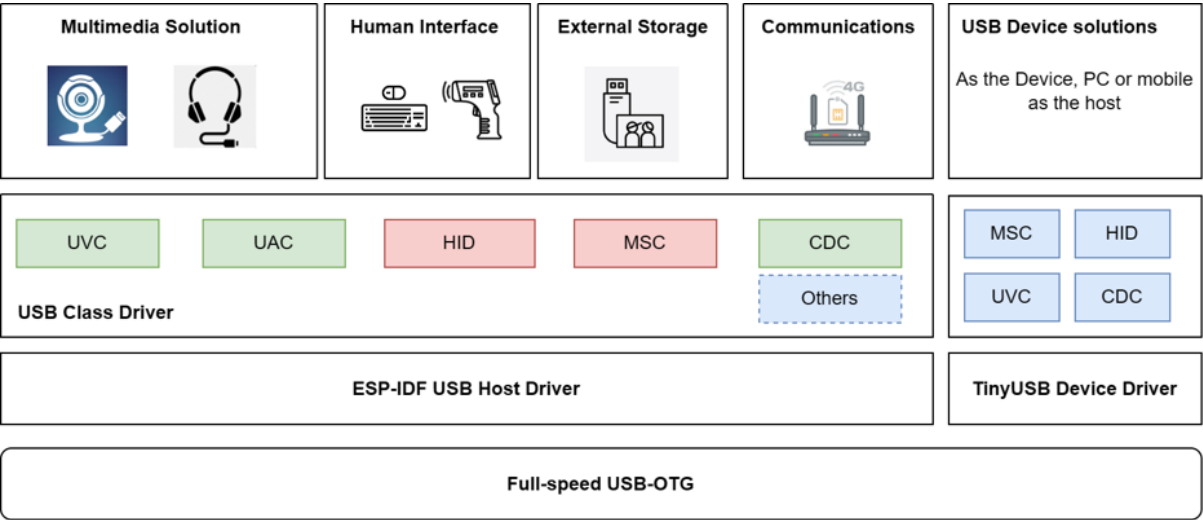
5.1.1 ESP USB 外设介绍

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/P4 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-OTG High-speed 控制器简介 USB OTG High-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 High-speed (480Mbps)，Full-speed (12Mbps) 和 Low-speed (1.5Mbps) 三种速率，支持 USB 2.0 协议。

ESP-IDF 从 v5.5 开始，ESP32-P4 原本独立的 USB PHY 实现已合并至与 S2/S3 共用的 PHY 模块，统一驱动结构，简化维护。

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。

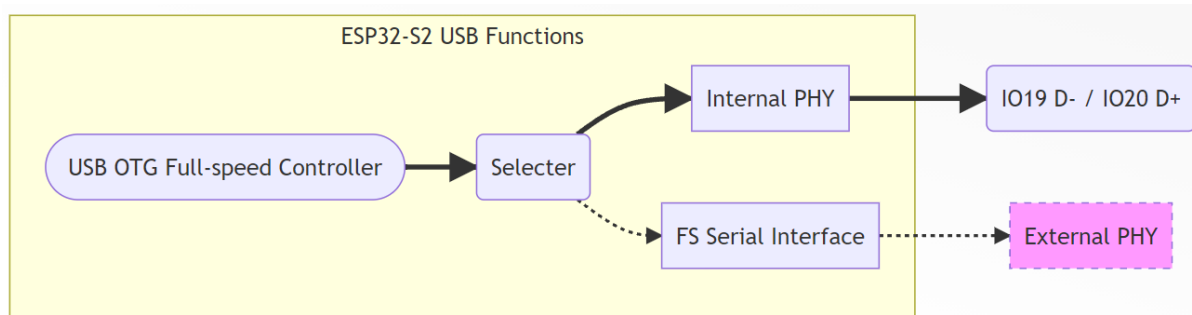
更多介绍，请参考[USB-PHY 介绍](#)。

ESP32-S/P/C 系列 USB 外设支持情况

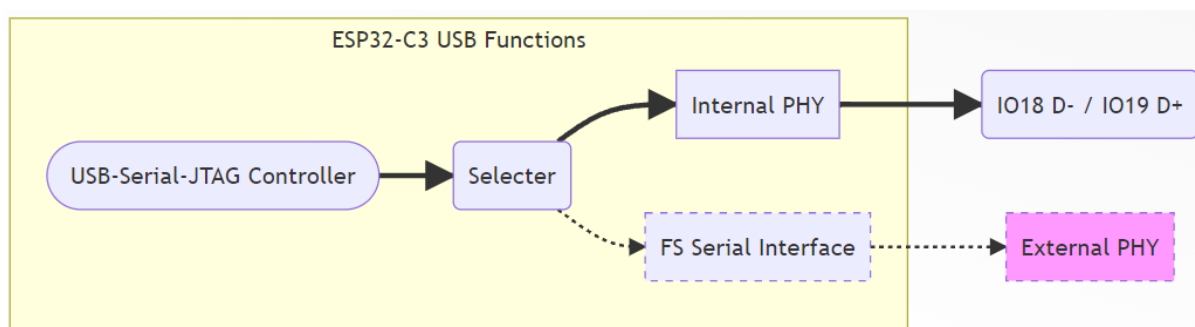
	USB OTG High-speed	USB OTG Full-speed	USB-Serial-JTAG	Full-speed PHY	High-speed PHY
ESP32-P4	√	√	√	√	√
ESP32-S3	X	√	√	√	X
ESP32-S2	X	√	X	√	X
ESP32-C6	X	X	√	√	X
ESP32-C3	X	X	√	√	X
ESP32-C2	X	X	X	X	X
ESP32	X	X	X	X	X
ESP8266	X	X	X	X	X

- √ : Supported
- X : Not Supported

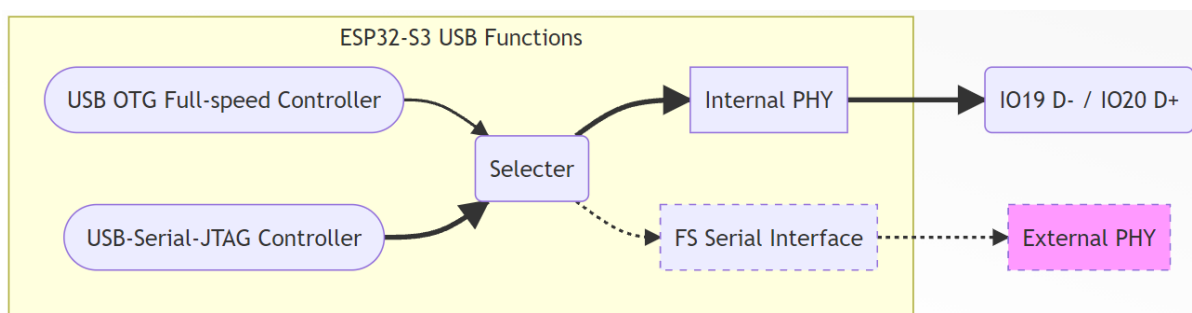
ESP32-S2 USB 功能简介 ESP32-S2 内置 **USB OTG Full-speed Controller** 和 **USB Full-speed PHY**，内部结构如下：



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：

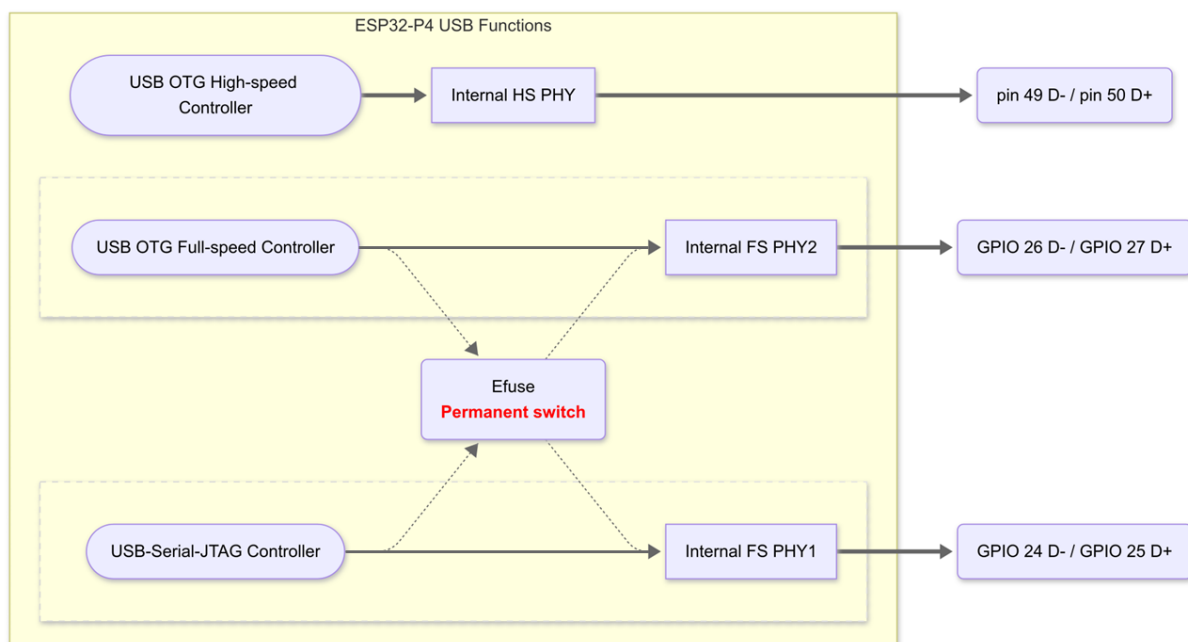


ESP32-S3 USB 功能简介 ESP32-S3 内置两个 USB 控制器，分别是 **USB OTG Full-speed Controller** 和 **USB-Serial-JTAG Controller**，内置一个 **USB Full-speed PHY**。内部 USB PHY 默认连接到 **USB-Serial-JTAG** 控制器，可通过烧写 eFuse 修改默认，或配置寄存器动态切换，也可通过增加外部 PHY，同时启用两个控制器。内部 USB PHY 的切换详情，参考[USB PHY 切换](#)。



ESP32-P4 USB 功能介绍 ESP32-P4 内置了三个 USB 控制器，分别是 **USB OTG High-speed Controller**、**USB OTG Full-speed Controller** 和 **USB-Serial-JTAG Controller**，内置一个 **USB High-speed PHY** 与两个 **USB Full-speed PHY**。默认情况下，FS_PHY1 连接到 USB 串行/JTAG 控制器，FS_PHY2 连接到 OTG_FS。用户可以通过 EFUSE_USB_PHY_SEL 更改连接关系。

- 0: FS_PHY1 连接 USB 串行/JTAG 控制器，FS_PHY2 连接 OTG_FS。
- 1: FS_PHY2 连接 USB 串行/JTAG 控制器，FS_PHY1 连接 OTG_FS。



5.1.2 USB-OTG 外设介绍

ESP32-S2/S3/P4 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/P4 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

对于 ESP32-P4 USB-OTG High Speed 总线传输速率为 480 Mbps，与 Full Speed 相比，在传输的最大尺寸、每毫秒传输包数量与理论有效速率上有所不同，如下表所示：

传输类型	控制	中断	批量	同步
传输的最大尺寸	64 字节	1024 字节	512 字节	1024 字节
每微帧传输包数量	•	6	13	7
理论有效速率	•	49152000 Bytes/s	53248000 Bytes/s	57344000 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 * 每微帧传输包数量 * 8000
- 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/P4 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](#)。

动态切换 USB-OTG 外设模式

USB-OTG 外设支持动态切换 USB-OTG 外设模式，用户可以通过动态的注册 USB Host Driver 或 USB Device Driver，实现 USB-OTG 外设的动态切换。

- 示例代码: USB OTG 手动切换 [usb/otg/usb_host_device_mode_manual_switch](#)

使用 USB-OTG 主动断开与主机的连接

可以通过将 USB OTG VBUSVALID 信号拉低，主动断开与主机的连接，从而实现 USB OTG 外设的断开。注意如果需要重新挂载设备，需要至少等待 10ms。请参考 [USB Device Self Power](#)

5.1.3 USB-Serial-JTAG 外设介绍

ESP32-S3/C3/P4 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。

若用户希望再次使用 USB-Serial-JTAG 功能，请参考如下代码：

```
#include "soc/soc_caps.h"
#include "soc/usb_serial_jtag_reg.h"
#include "hal/usb_serial_jtag_ll.h"

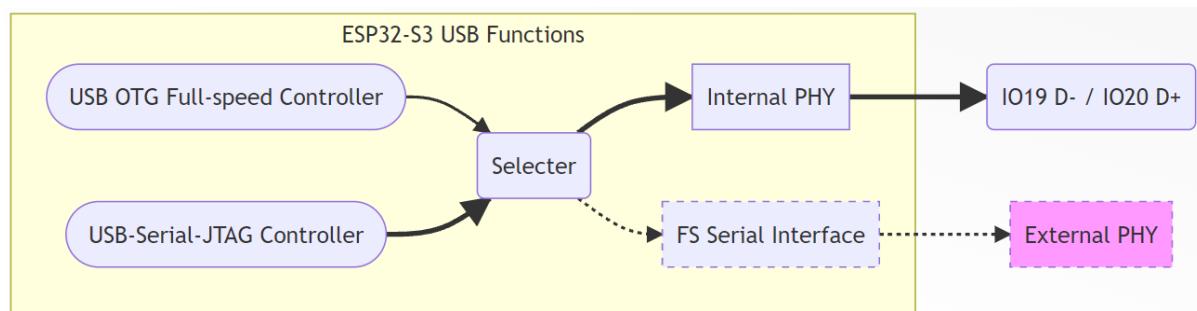
SET_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_PAD_PULL_
↪OVERWRITE);
CLEAR_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_DP_PULLUP);
SET_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_DP_PULLDOWN);
vTaskDelay(pdMS_TO_TICKS(10));
#if USB_SERIAL_JTAG_LL_EXT_PHY_SUPPORTED
usb_serial_jtag_ll_phy_enable_external(false); // Use internal PHY
usb_serial_jtag_ll_phy_enable_pad(true); // Enable USB PHY pads
#else // USB_SERIAL_JTAG_LL_EXT_PHY_SUPPORTED
usb_serial_jtag_ll_phy_set_defaults(); // External PHY not supported.↪
↪Set default values.
#endif // USB_WRAP_LL_EXT_PHY_SUPPORTED
CLEAR_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_DP_PULLDOWN);
SET_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_DP_PULLUP);
CLEAR_PERI_REG_MASK(USB_SERIAL_JTAG_CONF0_REG, USB_SERIAL_JTAG_PAD_PULL_
↪OVERWRITE);
```

5.1.4 USB PHY/Transceiver 介绍

USB 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/P4 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。同时，ESP32-P4 还内置 USB High-Speed PHY。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

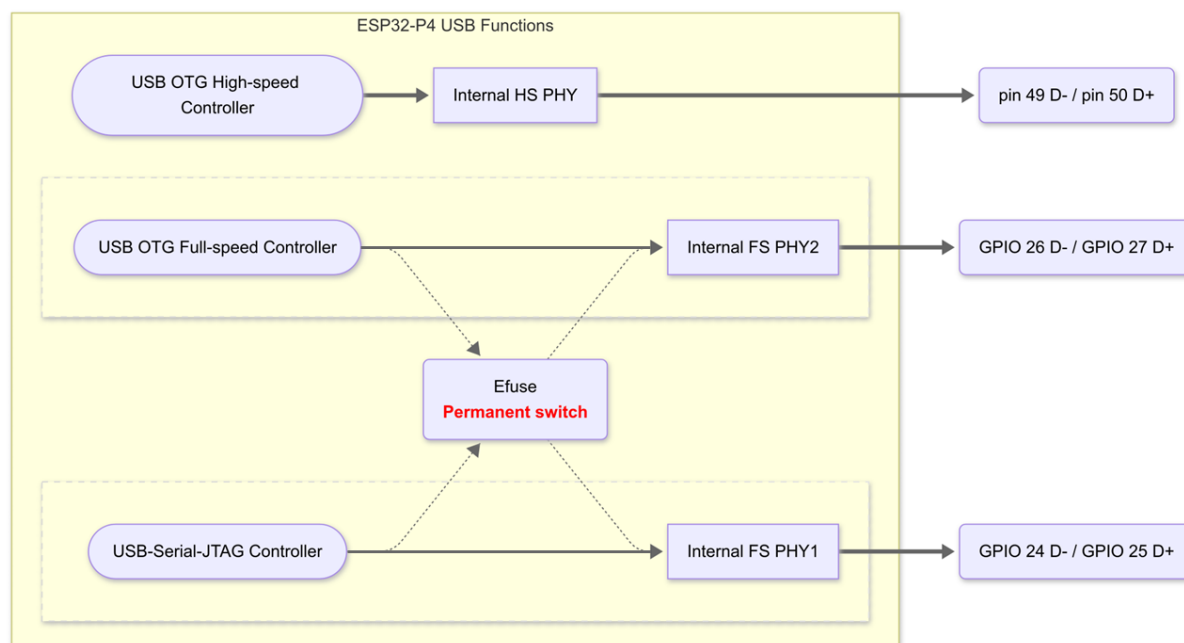
ESP32-S2/S3/C3 内部集成了 USB PHY，因此无需外接 PHY 芯片，可以直接与外部 USB 主机或设备通过 USB D+/D- 连接。但对于集成两个 USB 控制器的芯片，例如 ESP32-S3 内置 USB-OTG 和 USB-Serial-JTAG，两者共用一个内部 PHY，同一时间只能有一个工作。



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

对于 ESP32-P4 而言，内部集成了两个 USB Full-speed PHY 与一个 USB High-speed PHY。



内部 USB-PHY 对应的 GPIO，如下表所示：

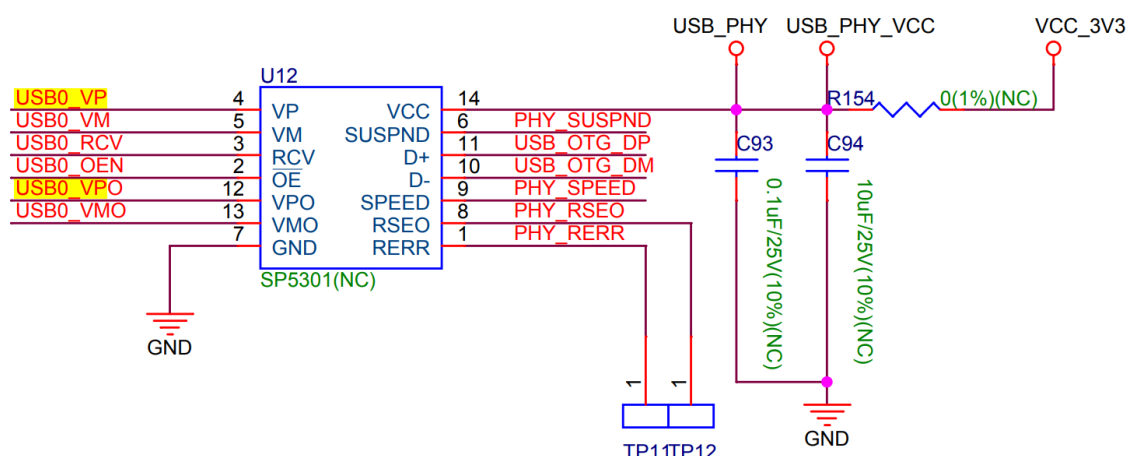
	D+	D-
ESP32-P4 FS_PHY1	25	24
ESP32-P4 FS_PHY2	27	26
ESP32-P4 HS_PHY	pin 50	pin 49

默认情况下，FS_PHY1 连接到 USB 串行/JTAG 控制器，FS_PHY2 连接到 OTG_FS。用户可以通过 EFUSE_USB_PHY_SEL 更改连接关系。

使用外部 PHY (ESP32-S2/S3)

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](#)。

此外，关于 USB 信号指令测试的详细步骤与固件，请参考 [USB Signal Quality Test](#)

5.1.6 USB Host 方案

ESP32-S2/S3/P4 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/P4 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 (Full-Speed, ESP32-S2/S3) 和 1080*1920@60fps (High-Speed, ESP32-P4)。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](#)

对于 ESP32-P4，请使用 `usb_host_uvc` 组件，支持 MJPEG、YUY2、H264 与 H265 格式，

链接:

- [usb_stream 组件](#)
- [usb_host_uvc 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, ESP32-P4
- 外设: USB-OTG
- USB 音频设备: 支持 PCM 格式

链接:

- [usb_stream 组件](#)
- [usb_host_uac 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, ESP32-P4
- 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, ESP32-P4
- 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](#)

ESP USB Hub 方案

支持通过 USB Hub 连接多个 USB 设备, 实现多设备同时工作。适用于多 USB 设备协同工作, 例如双摄像头视频采集、音视频同步处理、外设扩展与数据存储等。

特性:

- 支持通过 USB Hub 连接多个 USB 设备
- 支持热插拔

硬件：

- 芯片：ESP32-S2, ESP32-S3, ESP32-P4
- 外设：USB-OTG

链接：

- [USB Hub 双摄 Demo](#)

5.1.7 USB Device 方案

ESP32-S2/S3/P4 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, ESP32-P4
- 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, ESP32-P4
- 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, ESP32-P4
- 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, ESP32-P4
- 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](#)

USB 扩展屏方案

USB 扩展屏方案通过 USB 总线将设备作为一块扩展副屏使用。支持通过一根 USB 数据线传输音频, 触摸信息, 视频图像等数据。可以应用在电脑屏幕, 签字屏, 扩展屏等应用场景。

上位机驱动 USB2.0 驱动不支持 HDMI 传输, 因此需要上位机将图像数据传输给设备, 本应用方案暂时仅支持 Windows 平台, 采用 Windows 驱动 [IDD](#), 通过驱动可以拿到 Windows 的桌面图像, 因为 USB 速率限制, 在驱动中会先将图像处理成 JPEG 等格式, 然后通过 USB vendor 接口将图像传输到设备。每一帧图像添加 16 字节的头, 包含图像的宽高, 图像的格式, 图像的长度, 图像的压缩类型等。

特性:

- 支持通过 USB 传输图像
- 支持通过 USB 传输音频
- 支持通过 USB 传输触摸信息

硬件:

- 芯片: ESP32-S2, ESP32-S3, ESP32-P4
- 外设: USB-OTG

链接:

- [P4 USB 扩展屏](#)

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/P4 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 Ω) 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/P4 has no 5V_
↳tolerate pin.
 *
 *      5V VBUS ?????????? ?????????? GND
 *      ?????????? 100K ??????? 100K ?????????
 *
 *      ?????????? ? ??????????
 *
 *      ?
 *
 *      ?????????????????????
 *
 *      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:

1. 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 100KΩ）与 ESP32S2/S3 相连；
2. 将用于检测 VBUS 的 IO 初始化为 GPIO 输入模式；
3. 直接将 IO 配置到 tinyusb_config_t 中（详情可参考）:

```

#define VBUS_MONITORING_GPIO_NUM GPIO_NUM_4
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,
    .vbus_monitor_io = VBUS_MONITORING_GPIO_NUM,
};
ESP_ERROR_CHECK(tinyusb_driver_install(&tusb_cfg));

```

若使用原生 TinyUSB 开发，则需要 Phy 初始化阶段进行配置:

```

const usb_phy_otg_io_conf_t otg_io_conf = USB_PHY_SELF_POWERED_DEVICE(VBUS_
↳MONITORING_GPIO_NUM);
phy_conf.otg_io_conf = &otg_io_conf;
usb_new_phy(&phy_conf, &s_phy_hdl);

```

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	ESP32C5	ESP32C6	ESP32C61	ESP32H2	ESP32P4
Interface	usb-otg	usb-serial-jtag	usb-serial-jtag	usb-serial-jtag	usb-serial-jtag	usb-serial-jtag	usb-serial-jtag	usb-serial-jtag
VID	0x303a	0x303a	0x303a	0x303a	0x303a	0x303a	0x303a	0x303a
PID	0x0002	0x1001	0x1001	0x1001	0x1001	0x1001	0x1001	0x1001
Revision (bcdDevice)	N/A	v1.01	v1.01	v1.02	v1.02	v1.02	v1.02	v1.02
Serial	0	MAC 地址字符串	MAC 地址字符串	MAC 地址字符串	MAC 地址字符串	MAC 地址字符串	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)、或其他支持 usb-serial-jtag 接口的芯片，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
REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags\303A10010102 /
↪V IgnoreHWSerNum /t REG_BINARY /d 01
```

用户也可下载 ignore_hwserial_usb_serial_jtag.bat 脚本，右键选择管理员方式运行。

备注： 以上指令仅适用于 Windows 10/11，对于 Windows 7 或其它版本，请下载使用 ignore_hwserial_usb_serial_jtag_win7.bat 脚本，右键选择管理员方式运行。

5.1.10 USB 信号质量测试

USB 2.0 接口的信号质量测试是确保 USB 2.0 设备符合标准、获得 USB 2.0 认证标志的重要环节。信号质量测试内容主要包括：眼图测试、信号速率、结束域（EOP）位宽、交叉点电压范围、JK 对抖动、KJ 对抖动、连续抖动、上升时间和下降时间等。其中，眼图测试是串行数据应用中最基础且关键的测试项目之一。

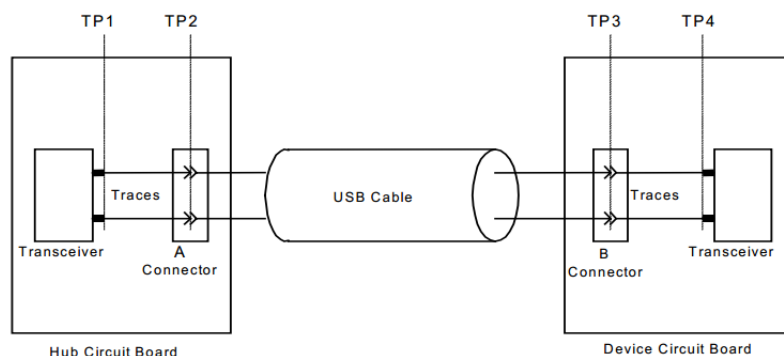


图 1: USB 2.0 高速信号测量平面

眼图测试

USB 2.0 High-Speed 信号的眼图测试结果需满足 USB-IF 协会制定的标准眼图模板要求。

USB 2.0 High-Speed 信号的眼图测试结果需符合 USB 2.0 规范的眼图模板：

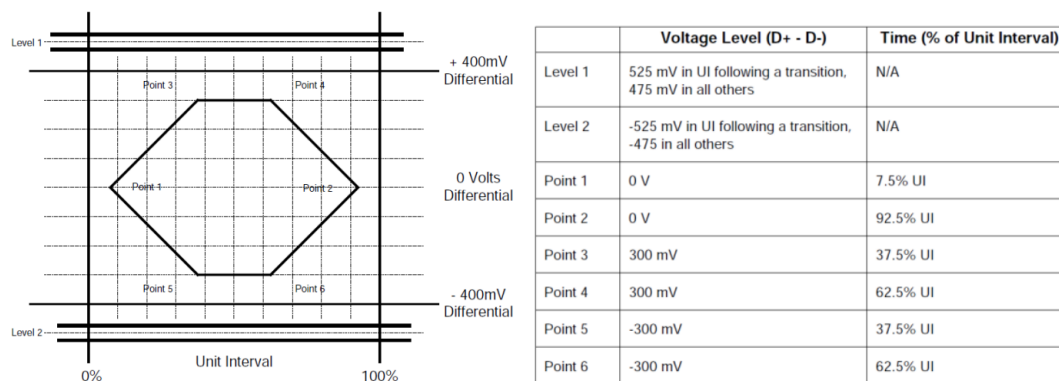


图 2: 在 TP2 处测量的集线器和在 TP3 处测量的设备（无固定电缆）的传输波形要求

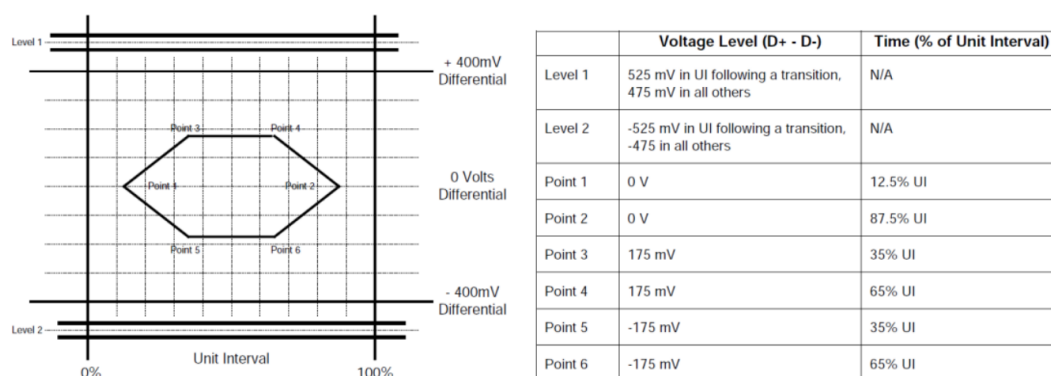


图 3: 在 TP2 处测量的设备（带有固定电缆）的传输波形要求

必备材料

- Windows PC
- USB XHSETT: [USB-IF 提供的测试工具](#)，用于控制 USB 2.0 设备和主机以及 USB 3.1 主机和集线器进入合规测试模式
- USB 治具：由仪器厂商提供或从 USB-IF 推荐的测试机构购买

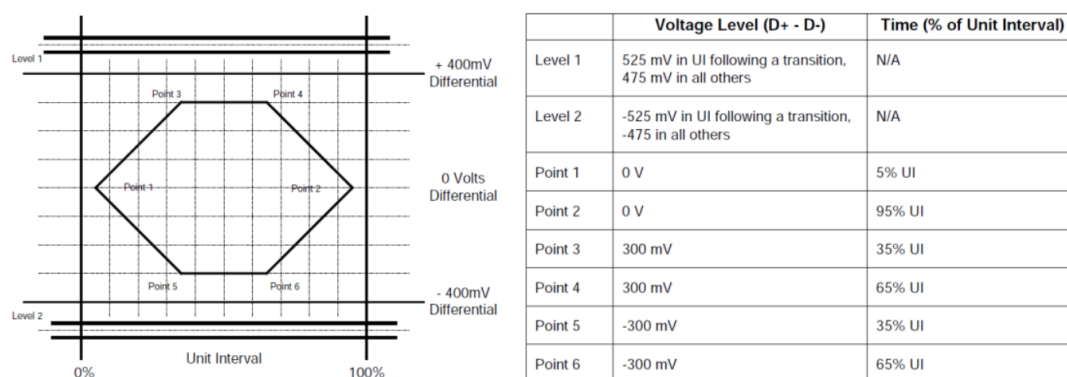


图 4: 在 TP1 处测量的集线器收发器和在 TP4 处测量的设备收发器的传输波形要求

- 示波器 (> 2 GHz)、一致性测试软件、探头等

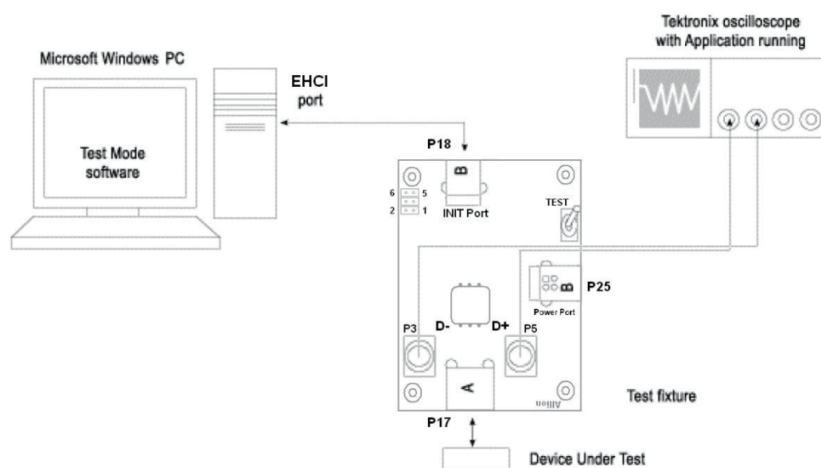


图 5: USB 2.0 信号质量测试系统

测试步骤

1. ESP32-S2/S3/P4 USB 测试固件 基于 [Launchpad](#) 平台，下载 ESP32-S2/S3/P4 USB 测试固件，其中 ESP32-P4 用于测试 High-Speed，ESP32-S2/S3 用于测试 Full-Speed。

2. 硬件连接 连接 PC、示波器与 USB 测试治具：High-Speed 与 Full/Low-Speed 测试治具不同，请参考示波器一致性测试软件上的连接示意图。

3. XHSETT 安装与设置 查询 PC 设备管理器中的 USB 端口信息：

- 如果 PC 为 USB 3.x 控制器，请使用 XHSETT 版本。xHCI 支持 USB 1.x/2.0/3.x
- 如果 PC 为 USB 2.0 控制器，请使用 HSETT 版本。HSETT 支持 USB 1.x/2.0

备注：HSETT/XHSETT 软件安装并启动后，会占用当前 PC 上的所有 USB 端口，导致 USB 端口暂时无法正常使用。因此，在测试主机上运行 HSETT/XHSETT 软件并重启 PC 之前，请务必提前连接好 PS/2 键盘和鼠标，确保操作不中断。如果无法使用 PS/2 鼠标，也可以考虑通过远程桌面等远程控制工具进行操作。

在完成 HSETT/XHSETT 软件安装后，请按照如下步骤进行设置：



图 6: High-Speed Device 信号质量测试连接示意图

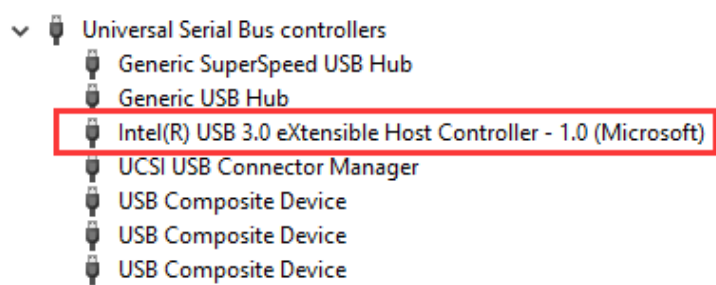


图 7: USB 3.x 控制器

打开 XHSETT 软件，选择 Device 模式：

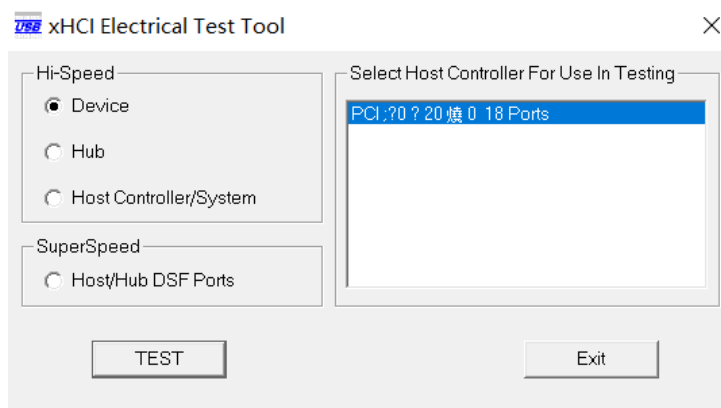


图 8: XHSETT 模式选择

点击 Enumerate Bus 按钮进行搜索，搜索到设备后选中待测设备并选择 Device Command：

- 对于 High-Speed 设备，选择 Device Command 中的 TEST_PACKET
- 对于 Full-Speed 设备，选择 Device Command 中的 (LOOP) DEVICE DESCRIPTOR

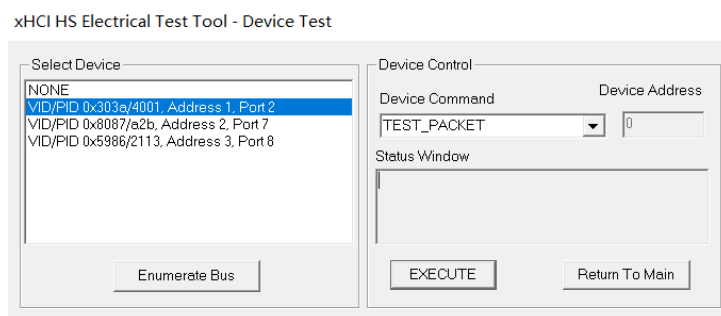


图 9: XHSETT 设备选择

4. 示波器眼图测试 USB 2.0 一致性分析软件通过对接口所发出的标准信号（XHSETT/HSETT）进行分析并生成眼图。各示波器厂商的一致性测试软件操作可能有所不同，请参考示波器厂商的操作说明进行眼图测试。

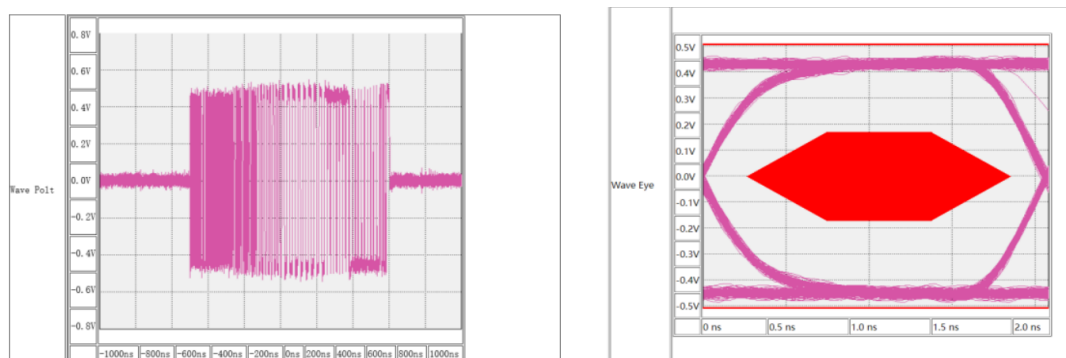


图 10: 示波器眼图测试

在 USB 2.0 眼图测试中，眼图张开得越大，信号质量越好。测试结果应完全落在眼图模板的合格区域内，任何触及或压到模板边界的情况都表示信号质量不达标。

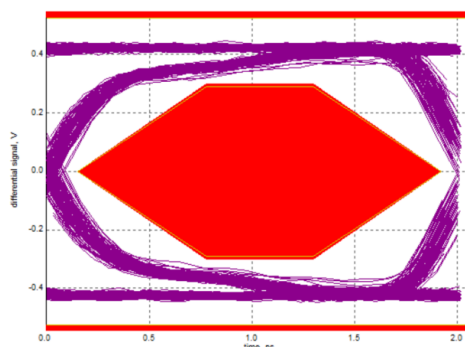


图 11: 可通过测试的眼图

4.1 USB 2.0 High-Speed 对于未通过测试的眼图，可从以下几个方面进行分析：

- 检查探头是否校准
- 使用规范的测试治具
- 替换线缆/连接器，排除劣质配件
- 检查 PCB 设计是否考虑 USB 2.0 规范，若在 USB High-Speed D+/D- 上使用了带有较大寄生电容的 ESD，眼图可能会测试失败，建议选择寄生电容低于 1 pF 的 ESD 二极管

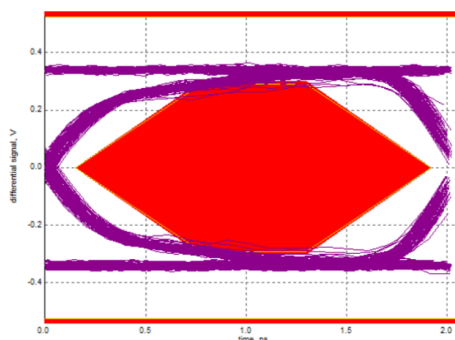


图 12: 未通过测试的眼图

4.2 USB 2.0 Full-Speed 对于 Full-Speed 设备的眼图测试，请参考示波器厂商提供的 USB 一致性测试软件进行测试：

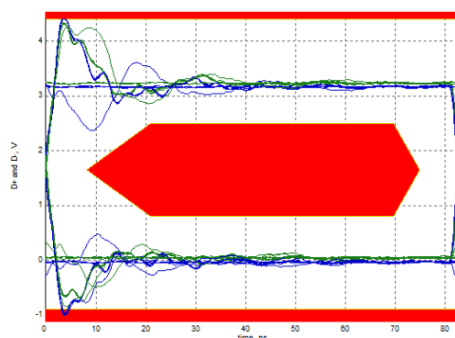


图 13: 未通过测试的 USB 2.0 Full-Speed 眼图（过冲）

对于上述 Full-Speed 未通过测试的眼图，需在 D+/D- 串接电阻以解决过冲。实测结果表明，串接 22 Ω 、33 Ω 或 44 Ω 电阻均能改善过冲，其中以 33 Ω 的效果最佳。

5.1.11 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。

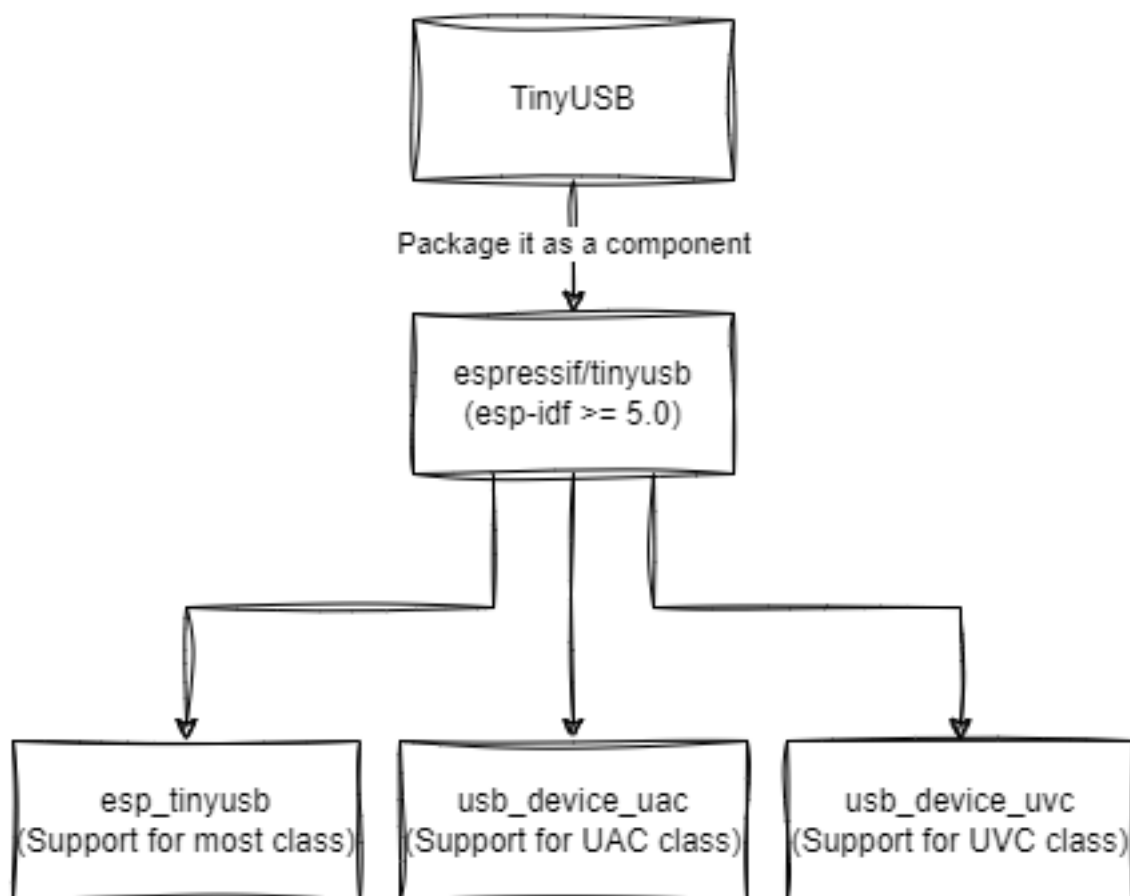


图 14: TinyUSB Components

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。主要是补充了 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
- **ASYNCR 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 ASYNCR 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/M420/M20T	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](#)

5.1.12 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)
```

- **CFG_TUSB_RHPORT1_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 高速。

```
#define CFG_TUSB_RHPORT1_MODE    (OPT_MODE_DEVICE | OPT_MODE_HIGH_
↪ 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_TUSB_DEBUG_PRINTF**: 用于定义 tinyusb 的 log 打印函数。

```
#define CFG_TUSB_DEBUG_PRINTF    esp_rom_printf
```

- **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/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 \(ESP32-S2/S3\)](#)

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;
    }
    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_frequence = 16000, //microphone frequency, Hz
    .spk_bit_resolution = 16, //speaker resolution, bits
    .spk_samples_frequence = 16000, //speaker frequency, Hz
    .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 $\geq$ 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))) {
        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_frequency equals 0, which means the frequency can be set to any value between samples_frequency_min and samples_frequency_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_frequence`
sample frequency

`uint32_t samples_frequence_min`
min sample frequency

`uint32_t samples_frequence_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/ESP32-P4 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

bool **skip_msc_connect_wait**

Skip waiting for MSC device to connect, if true means MSC device is already connected

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_usbh_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 `usbh_cdc_driver_install` 配置, 用户可以配置好 USB CDC 驱动程序, 并通过设置 `skip_init_usb_host_driver` 配置项, 在组件内部初始化好 USB HOST Driver 协议栈。

```
/* 安装 USB CDC 驱动, 并由驱动内部初始化好 USB Host Driver 协议栈 */
usbh_cdc_driver_config_t config = {
    .driver_task_stack_size = 1024 * 4,
    .driver_task_priority = 5,
    .xCoreID = 0,
    .skip_init_usb_host_driver = false,
    .new_dev_cb = cdc_new_dev_cb,
};
```

2. 使用 `usbh_cdc_create` 配置, 用户可以简单配置 `itf_num` 接口号和内部 `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 `connect disconnect recv_data`:

```
/* 安装 USB 主机 CDC 驱动程序, 配置 bulk 端点地址和内部 ringbuffer 的大小 */
usbh_cdc_device_config_t dev_config = {
    .vid = 0,
    .pid = 0,
    .itf_num = 1,
    /* 如果值为非 0, 则启用内部 ringbuffer */
    .rx_buffer_size = 2048,
    .tx_buffer_size = 2048,
    .cbs = {
        .connect = cdc_connect_cb,
        .disconnect = cdc_disconnect_cb,
        .user_data = NULL
    },
};

usbh_cdc_handle_t handle = NULL;
usbh_cdc_create(&dev_config, &handle);
/* 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 = 3;
usbh_cdc_handle_t handle2 = NULL;
usbh_cdc_create(&dev_config, &handle2);
#endif
```

3. 驱动程序初始化后, 内部状态机将自动处理 USB 的热插拔。
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 `usbh_cdc_get_rx_buffer_size` 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 `usbh_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_delete` 可以删除 USB CDC 设备, 释放掉 `ringbuffer` 以及其他资源。
7. `usbh_cdc_driver_uninstall` 可以完全卸载 USB 驱动程序以释放所有资源。

不使用内部 ringbuffer

如不想使用内部 `ringbuffer`, 可以设置 `rx_buffer_size` 和 `tx_buffer_size` 为 0。

```
/* 安装 USB 主机 CDC 驱动程序, 配置 bulk 端点地址和内部 ringbuffer 的大小 */
usbh_cdc_device_config_t dev_config = {
    .vid = 0,
    .pid = 0,
    .itf_num = 1,
    /* 如果值为 0, 则不启用内部 ringbuffer */
```

(下页继续)


```

.rx_buffer_size = 0,
.tx_buffer_size = 0,
.cbs = {
    /* 配置接收回调函数 */
    .recv_data = cdc_recv_data_cb,
    .connect = cdc_connect_cb,
    .disconnect = cdc_disconnect_cb,
    .user_data = NULL
},
};

usbh_cdc_handle_t handle = NULL;
usbh_cdc_create(&dev_config, &handle);
/* 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 = 3;
usbh_cdc_handle_t handle2 = NULL;
usbh_cdc_create(&dev_config, &handle2);
#endif

```

1. 应配置 `recv_data` 回调函数，当有数据接收时，会调用该回调函数。
2. 在回调函数中调用 `usbh_cdc_get_rx_buffer_size` 获取数据大小，然后调用 `usbh_cdc_read_bytes` 读取数据。注意 `ticks_to_wait` 参数应设置为 0，不支持堵塞读取。
3. 如果需要发送数据，可以调用 `usbh_cdc_write_bytes` 发送数据，通过配置 `ticks_to_wait` 参数，可以设置等待时间，如果等待超时，将返回 `ESP_ERR_TIMEOUT` 错误。

示例代码

[usb/host/usb_cdc_basic](#)

API 参考

Header File

- [components/usb/iot_usb_cdc/include/iot_usb_cdc.h](#)

Functions

`esp_err_t usbh_cdc_driver_install` (const [usbh_cdc_driver_config_t](#) *config)

Install the USB CDC driver.

This function installs and initializes the USB CDC driver. It sets up internal data structures, creates necessary tasks, and registers the USB host client. If the USB host driver is not already initialized, it will create and initialize it.

The function performs the following steps:

- a. Verifies input arguments and state.
- b. Optionally initializes the USB host driver.
- c. Allocates memory for the USB CDC object.
- d. Creates necessary synchronization primitives (event group, mutex).
- e. Creates the driver task.
- f. Registers the USB host client with the provided configuration.
- g. Initializes the CDC driver structure and resumes the driver task.

In case of an error, the function will clean up any resources that were allocated before the failure occurred.

备注: If the `skip_init_usb_host_driver` flag in the config is set to true, the function will skip the USB host driver initialization.

参数 **config** –Pointer to a configuration structure (`usbh_cdc_driver_config_t`) that specifies the driver's settings such as task stack size, priority, core affinity, and new device callback.

返回

- `ESP_OK`: Driver installed successfully
- `ESP_ERR_INVALID_ARG`: Null configuration parameter or invalid arguments
- `ESP_ERR_INVALID_STATE`: USB CDC driver already installed
- `ESP_ERR_NO_MEM`: Memory allocation failed
- `ESP_FAIL`: Failed to create necessary tasks or USB host installation failed

`esp_err_t usbh_cdc_driver_uninstall` (void)

Uninstall the USB CDC driver.

This function uninstalls the USB CDC driver, releasing any allocated resources and stopping all driver-related tasks. It ensures that all CDC devices are closed before proceeding with the uninstallation.

返回

- `ESP_OK`: Driver uninstalled successfully
- `ESP_ERR_INVALID_STATE`: Driver is not installed or devices are still active
- `ESP_ERR_NOT_FINISHED`: Timeout occurred while waiting for the CDC task to finish

`esp_err_t usbh_cdc_create` (const `usbh_cdc_device_config_t` *config, `usbh_cdc_handle_t` *cdc_handle)

Create a new USB CDC device handle.

This function allocates and initializes a new USB CDC device based on the provided configuration, including setting up the ring buffers for data transmission and reception, and searching for and opening the appropriate USB device.

参数

- **cdc_handle** –[out] Pointer to a variable that will hold the created CDC device handle upon successful creation
- **config** –[in] Pointer to a `usbh_cdc_device_config_t` structure containing the configuration for the new device, including vendor ID (VID), product ID (PID), and buffer sizes

返回

- `ESP_OK`: CDC device successfully created
- `ESP_ERR_INVALID_STATE`: USB CDC driver is not installed
- `ESP_ERR_INVALID_ARG`: Null `cdc_handle` or `config` parameter
- `ESP_ERR_NO_MEM`: Memory allocation failed or ring buffer creation failed
- `ESP_FAIL`: Failed to open the CDC device

`esp_err_t usbh_cdc_delete` (`usbh_cdc_handle_t` cdc_handle)

Delete a USB CDC device handle.

This function deletes the specified USB CDC device handle, freeing its associated resources, including ring buffers and semaphores.

备注: When deleting the device, please avoid writing or reading data.

参数 **cdc_handle** –[in] The CDC device handle to delete

返回

- `ESP_OK`: CDC device deleted successfully
- `ESP_ERR_INVALID_STATE`: USB CDC driver is not installed
- `ESP_ERR_INVALID_ARG`: Invalid CDC handle provided

```
esp_err_t usbh_cdc_send_custom_request (usbh_cdc_handle_t cdc_handle, uint8_t bmRequestType,
                                         uint8_t bRequest, uint16_t wValue, uint16_t wIndex,
                                         uint16_t wLength, uint8_t *data)
```

Send a custom control request to the USB CDC device.

This function sends a custom control request to the USB CDC device with the specified parameters. It can be used for both IN and OUT transfers.

参数

- **cdc_handle** –[in] The CDC device handle
- **bmRequestType** –[in] The request type bitmask
- **bRequest** –[in] The request ID
- **wValue** –[in] The request value
- **wIndex** –[in] The request index
- **wLength** –[in] The length of data to transfer
- **data** –[inout] Pointer to the data buffer:
 - For OUT transfers: data to be sent to device
 - For IN transfers: buffer to store received data

返回

- ESP_OK: Request sent successfully
- ESP_ERR_INVALID_ARG: Invalid arguments (NULL handle/data buffer)
- ESP_ERR_INVALID_STATE: Device is not connected
- ESP_ERR_TIMEOUT: Control transfer timed out
- ESP_ERR_INVALID_RESPONSE: Control transfer failed

```
esp_err_t usbh_cdc_write_bytes (usbh_cdc_handle_t cdc_handle, const uint8_t *buf, size_t length,
                                TickType_t ticks_to_wait)
```

Write data to the USB CDC device.

With internal ring buffer, this function writes data to the specified USB CDC device by pushing the data into the output ring buffer. If the buffer is full or the device is not connected, the write will fail.

Without internal ring buffer, this function writes data to the specified USB CDC device by sending data directly to the device. Please set the ticks_to_wait for blocking write.

参数

- **cdc_handle** –[in] The CDC device handle
- **buf** –[in] Pointer to the data buffer to write
- **length** –[in] Pointer to the length of data to write. On success, it remains unchanged, otherwise set to 0.
- **ticks_to_wait** –[in] The maximum amount of time to wait for the write operation to complete

返回

- ESP_OK: Data written successfully
- ESP_ERR_INVALID_ARG: Invalid argument (NULL handle, buffer, or length)
- ESP_ERR_INVALID_STATE: Device is not connected

```
esp_err_t usbh_cdc_read_bytes (usbh_cdc_handle_t cdc_handle, uint8_t *buf, size_t *length, TickType_t
                               ticks_to_wait)
```

Read data from the USB CDC device.

With internal ring buffer, this function reads data from the specified USB CDC device by popping data from the input ring buffer. If no data is available or the device is not connected, the read will fail.

Without internal ring buffer, this function reads data from the specified USB CDC device by receiving data directly from the device. You can call this function in the recv_data callback function. Please set ticks_to_wait to zero for non-blocking read.

参数

- **cdc_handle** –[in] The CDC device handle
- **buf** –[out] Pointer to the buffer where the read data will be stored

- **length** –[inout] Pointer to the length of data to read. On success, it is updated with the actual bytes read.
- **ticks_to_wait** –[in] The maximum amount of time to wait for the read operation to complete

返回

- ESP_OK: Data read successfully
- ESP_FAIL: Failed to read data
- ESP_ERR_INVALID_ARG: Invalid argument (NULL handle, buffer, or length)
- ESP_ERR_INVALID_STATE: Device is not connected

esp_err_t **usbh_cdc_flush_rx_buffer** (*usbh_cdc_handle_t* cdc_handle)

Flush the receive buffer of the USB CDC device.

This function clears the receive buffer, discarding any data currently in the buffer.

Not supported for no internal ring buffer mode.

参数 **cdc_handle** –[in] The CDC device handle

返回

- ESP_OK: Receive buffer flushed successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle provided
- ESP_ERR_NOT_SUPPORTED: Not supported for no internal ring buffer mode

esp_err_t **usbh_cdc_flush_tx_buffer** (*usbh_cdc_handle_t* cdc_handle)

Flush the transmit buffer of the USB CDC device.

This function clears the transmit buffer, discarding any data currently in the buffer.

Not supported for no internal ring buffer mode.

参数 **cdc_handle** –[in] The CDC device handle

返回

- ESP_OK: Transmit buffer flushed successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle provided
- ESP_ERR_NOT_SUPPORTED: Not supported for no internal ring buffer mode

esp_err_t **usbh_cdc_get_rx_buffer_size** (*usbh_cdc_handle_t* cdc_handle, size_t *size)

Get the size of the receive buffer of the USB CDC device.

This function retrieves the current size of the receive buffer in bytes.

For no internal ring buffer, you can call this function in the `recv_data` callback function.

参数

- **cdc_handle** –[in] The CDC device handle
- **size** –[out] Pointer to store the size of the receive buffer

返回

- ESP_OK: Size retrieved successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle provided

esp_err_t **usbh_cdc_register_new_dev_cb** (*usbh_cdc_new_dev_cb_t* new_dev_cb, void *user_data)

Register an additional callback function for new device detection.

This function registers a callback that will be invoked when a new USB CDC device is connected. The callback will be automatically unregistered when `usbh_cdc_driver_uninstall()` is called.

参数

- **new_dev_cb** –[in] Callback function to be called when a new device is connected
- **user_data** –[in] User data pointer that will be passed to the callback function

返回

- ESP_OK: Callback registered successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle or callback function
- ESP_ERR_NO_MEM: Failed to allocate memory for callback registration

esp_err_t **usbh_cdc_unregister_new_dev_cb** (*usbh_cdc_new_dev_cb_t* new_dev_cb)

Unregister the new device callback which same as the one registered by `usbh_cdc_register_new_dev_cb`.

参数 **new_dev_cb** –[in] The new device callback function to unregister

返回

- ESP_OK: Callback unregistered successfully
- ESP_ERR_INVALID_ARG: Invalid callback function

esp_err_t **usbh_cdc_get_state** (*usbh_cdc_handle_t* cdc_handle, *usbh_cdc_state_t* *state)

Get the connect state of given interface.

参数

- **cdc_handle** –[in] The CDC device handle
- **state** –[out] Pointer to store the connect state

返回

- ESP_OK: Size retrieved successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle provided

esp_err_t **usbh_cdc_desc_print** (*usbh_cdc_handle_t* cdc_handle)

Print the USB CDC device descriptors.

This function retrieves and prints the USB device and configuration descriptors for the specified USB CDC device. It checks that the device is properly connected and open before retrieving and printing the descriptors.

参数 **cdc_handle** –[in] The CDC device handle

返回

- ESP_OK: Descriptors printed successfully
- ESP_ERR_INVALID_ARG: Invalid CDC handle provided
- ESP_ERR_INVALID_STATE: Device is not connected or not yet open

Structures

struct **usbh_cdc_driver_config_t**

CDC driver configuration.

Public Members

size_t **task_stack_size**

Stack size of the driver's task

unsigned **task_priority**

Priority of the driver's task

int **task_coreid**

Core of the driver's task, Set it to -1 to not specify the core.

bool **skip_init_usb_host_driver**

Skip initialization of USB host driver

usbh_cdc_new_dev_cb_t **new_dev_cb**

Callback function when a new device is connected

void ***user_data**

Pointer to user data that will be passed to the callbacks

struct **usbh_cdc_event_callbacks_t**

Callback structure for CDC device events.

Public Members

usbh_cdc_event_cb_t **connect**

USB connect callback, set NULL if use

usbh_cdc_event_cb_t **disconnect**

USB disconnect callback, set NULL if not use

usbh_cdc_event_cb_t **recv_data**

USB receive data callback, set NULL if not use

usbh_cdc_notif_cb_t **notif_cb**

USB notification callback, set NULL if not use

void ***user_data**

Pointer to user data that will be passed to the callbacks

struct **usbh_cdc_config**

Configuration structure for initializing a USB CDC device.

Public Members

uint16_t **vid**

Vendor ID: If set, the `pid` parameter must be configured. If not set, it will default to opening the first connected device.

uint16_t **pid**

Product ID: If set, the `vid` parameter must be configured. If not set, it will default to opening the first connected device.

int **itf_num**

interface numbers

size_t **rx_buffer_size**

Size of the receive ringbuffer size, if set to 0, not use ringbuffer

size_t **tx_buffer_size**

Size of the transmit ringbuffer size, if set to 0, not use ringbuffer

usbh_cdc_event_callbacks_t **cbs**

Event callbacks for the CDC device

Macros

CDC_HOST_ANY_VID

CDC_HOST_ANY_PID

Type Definitions

typedef struct usbh_cdc_t ***usbh_cdc_handle_t**

typedef void (***usbh_cdc_new_dev_cb_t**)(usb_device_handle_t usb_dev, void *user_data)

New USB device callback.

Provides already opened usb_dev, that will be closed after this callback returns. This is useful for peeking device's descriptors, e.g. peeking VID/PID and loading proper driver.

Attention This callback is called from USB Host context, so the CDC device can't be opened here.

typedef void (***usbh_cdc_event_cb_t**)([usbh_cdc_handle_t](#) cdc_handle, void *user_data)

Callback structure for CDC device events.

Param cdc_handle [in] Handle to the CDC device that sent the notification

Param user_data [in] User data pointer that was passed during callback registration

typedef void (***usbh_cdc_notif_cb_t**)([usbh_cdc_handle_t](#) cdc_handle, [iot_cdc_notification_t](#) *notif, void *user_data)

Callback function for CDC device notifications.

This callback is invoked when a CDC device sends a notification event. The notification data is passed through the notif parameter.

Param cdc_handle [in] Handle to the CDC device that sent the notification

Param notif [in] Pointer to the notification data structure containing notification details

Param user_data [in] User data pointer that was passed during callback registration

typedef struct [usbh_cdc_config](#) **usbh_cdc_device_config_t**

Configuration structure for initializing a USB CDC device.

Enumerations

enum **usbh_cdc_state_t**

State of the USB CDC device.

Values:

enumerator **USBH_CDC_CLOSE**

enumerator **USBH_CDC_OPEN**

enumerator **USBH_CDC_STATE_MAX**

5.2.4 USB PPP

PPP 是一种网络中最为基础的协议。PPP 协议（Point-to-Point Protocol）是一种数据链路层协议，它是为在同等单元之间传输数据包这样的简单链路而设计的。这种链路提供全双工操作，并按照顺序传递数据包。PPP 为基于各种主机、网桥和路由器的简单连接提供一种共通的解决方案。

`iot_usbh_modem` 组件实现了 USB 主机 PPP 拨号的全流程。支持通过 USB 接口连接 4G Cat.1/4 模块，实现 PPP 拨号上网。支持通过 Wi-Fi softAP 热点共享互联网给其他设备。

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

已支持的模组型号：

下表是已经支持的 4G 模组型号，可直接在 `menuconfig` 中通过宏 `MODEM_TARGET` 配置，如配置后不生效，请直接在 `menuconfig` 中选择 `MODEM_TARGET_USER` 并手动配置 ITF 接口。

备注： 同一个模组可能存在多个功能固件，具体请咨询模组厂商是否支持 PPP 拨号。

模组型号
ML302-DNLM/CNLM
NT26
EC600NCNLC-N06
AIR780E
MC610_EU
EC20_CE
EG25_GL
YM310_X09
SIM7600E
A7670E
SIM7070G
SIM7080G

设置 PPP 接口

PPP 接口一般为 USB CDC 接口，一定具有一个 CDC data 接口，包含两个 bulk 端点，用于输入和输出数据。可能具有一个 CDC notify 接口，包含一个 interrupt 端点。本应用代码中并不使用该接口。

PPP 接口可用于传输 AT 指令和 PPP 数据传输，通过发送“+++”数据进行切换。

示例描述符 大部分的 PPP 接口的 `bInterfaceClass` 都为 `0xFF` (Vendor Specific Class)。如下图改 USB 设备的 PPP 接口的 `bInterfaceNumber` 为 `0x03`。

示例 USB 描述符：

备注： 以下描述符为示例，并非所有的描述符都如下，可能带有 IAD 接口描述符或者不具有 interrupt 端点。

```

*** Interface descriptor ***
bLength 9
bDescriptorType 4
bInterfaceNumber 3
bAlternateSetting 0
bNumEndpoints 3
bInterfaceClass 0xff
bInterfaceSubClass 0x0
bInterfaceProtocol 0x0
iInterface 8
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x8a    EP 10 IN
    bmAttributes 0x3        INT
    wMaxPacketSize 16
    bInterval 16
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x82    EP 2 IN
    bmAttributes 0x2        BULK
    wMaxPacketSize 64
    bInterval 0
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x1     EP 1 OUT
    bmAttributes 0x2        BULK
    wMaxPacketSize 64
    bInterval 0

```

当我们找到了 PPP 接口后，可以配置 MODEM_TARGET 为 *MODEM_TARGET_USER*，并配置 MODEM_USB_ITF 为 PPP 接口的 bInterfaceNumber。

双 PPP 接口 为了保证数据传输时还能传输 AT 指令，可以使用两个 PPP 接口，一个用于传输数据，一个用于传输 AT 指令。需要额外的配置 MODEM_USB_ITF2。

备注： 是否具有第二个 AT 指令接口，要视设备而定。

5.2.5 USB RNDIS 主机驱动

[iot_usbh_rndis](#) 是基于 USB 接口的 RNDIS 协议的主机驱动。

RNDIS 协议

RNDIS(Remote NDIS) 通过将 TCP/IP 封装在 USB 报文里，实现网络通信。

USB 描述符

RNDIS 设备默认自己为 USB 配置中的唯一功能。对于复合设备，RNDIS 希望自己是第一个 USB 配置，即 bInterfaceNumber 为 0 的接口。

RNDIS 接口一般由一个 IAD 描述符和两个接口描述符组成。IAD 描述符的 bInterfaceCount 为 2，bFunctionClass 为 0xe0。

示例：

```
*** Interface Association Descriptor ***
bLength 8
bDescriptorType 11
bFirstInterface 0
bInterfaceCount 2
bFunctionClass 0xe0
bFunctionSubClass 0x1
bFunctionProtocol 0x3
iFunction 5
    *** Interface descriptor ***
    bLength 9
    bDescriptorType 4
    bInterfaceNumber 0
    bAlternateSetting 0
    bNumEndpoints 1
    bInterfaceClass 0xe0
    bInterfaceSubClass 0x1
    bInterfaceProtocol 0x3
    iInterface 5
        *** Endpoint descriptor ***
        bLength 7
        bDescriptorType 5
        bEndpointAddress 0x85    EP 5 IN
        bmAttributes 0x3        INT
        wMaxPacketSize 16
        bInterval 16
    *** Interface descriptor ***
    bLength 9
    bDescriptorType 4
    bInterfaceNumber 1
    bAlternateSetting 0
    bNumEndpoints 2
    bInterfaceClass 0xa
    bInterfaceSubClass 0x0
    bInterfaceProtocol 0x0
    iInterface 5
        *** Endpoint descriptor ***
        bLength 7
        bDescriptorType 5
        bEndpointAddress 0x87    EP 7 IN
        bmAttributes 0x2        BULK
        wMaxPacketSize 64
        bInterval 0
        *** Endpoint descriptor ***
        bLength 7
        bDescriptorType 5
        bEndpointAddress 0x6     EP 6 OUT
        bmAttributes 0x2        BULK
        wMaxPacketSize 64
        bInterval 0
```

通信流程

1. 通过控制端点发送初始化消息 `REMOTE_NDIS_INITIALIZE_MSG`，设备会返回 `REMOTE_NDIS_INITIALIZE_CMPLT_MSG` 消息。
2. 接着主机需要发送 `REMOTE_NDIS_QUERY_MSG` 消息，用于查询对象描述符 (OID)，并根据 OID 来查询和支持设备。如拿到设备的 MAC 地址，连接状态，连接速度等。
3. 接着主机端就可以发起 DHCP 请求，获取 IP 地址。

帧格式

RNDIS 协议的帧格式如下：

发送：RNDIS 数据包需要在以太网帧前添加 RNDIS 数据头 `rndis_data_packet_t`，然后添加以太网帧报文。

接收：RNDIS 数据包在以太网帧头部添加 RNDIS 数据头 `rndis_data_packet_t`。需要先解析 RNDIS 数据头，再拿到以太网帧。

速率测试

ESP32-S3 连接 4G 网卡，并开始 softAP，手机连接 softAP 进行网络测速：

芯片	上行 (Mbps)	下行 (Mbps)
ESP32-S3	5.8	7.9

API 参考

Header File

- [components/usb/iot_usbh_rndis/include/iot_usbh_rndis.h](#)

Functions

`esp_err_t iot_eth_new_usb_rndis (const iot\_usbh\_rndis\_config\_t *config, iot_eth_driver_t **ret_handle)`

Create a new USB RNDIS Ethernet driver.

This function initializes a new USB RNDIS Ethernet driver with the specified configuration. It allocates memory for the driver, sets up the driver functions, and returns a handle to the driver.

参数

- **config** –Pointer to the RNDIS configuration structure.
- **ret_handle** –Pointer to a location where the handle to the new Ethernet driver will be stored.

返回

- `ESP_OK`: Driver created successfully
- `ESP_ERR_INVALID_ARG`: Invalid argument (NULL config or ret_handle)
- `ESP_ERR_NO_MEM`: Memory allocation failed

Structures

struct `iot_usbh_rndis_config_t`

USB Host Ethernet ECM Configuration.

Public Members

bool **auto_detect**

Auto detect RNDIS device

TickType_t **auto_detect_timeout**

Auto detect timeout in ticks, used when auto_detect is true

uint16_t **vid**

USB device vendor ID, used when auto_detect is false

uint16_t **pid**

USB device product ID, used when auto_detect is false

int **itf_num**

interface numbers, used when auto_detect is false

5.2.6 USB ECM 主机驱动

`iot_usbh_ecm` 是基于 USB 接口的 ECM 协议的主机驱动。

ECM 协议

ECM(Ethernet Control Model) 通过将以太网帧封装在 USB 报文里, 实现网络通信。

USB 描述符

ECM 接口一般由一个两个接口描述符和类特殊描述符组成。第一个接口描述符的 `bInterfaceClass` 为 `0x02`, `bInterfaceSubClass` 为 `0x06`。ECM 功能描述符 `usb_ecm_function_desc_t` 中的 `iMACAddress` 为 MAC 地址的字符串索引, `wMaxSegmentSize` 为以太网帧的最大长度 (默认为 1514)。

备注: 一些 ECM 设备的配置描述符不为默认的 `config 1`, 因此需要开启宏 `CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK` 并实现枚举过滤回调函数。

示例描述符:

```

*** Interface descriptor ***
bLength 9
bDescriptorType 4
bInterfaceNumber 0
bAlternateSetting 0
bNumEndpoints 1
bInterfaceClass 0x2
bInterfaceSubClass 0x6
bInterfaceProtocol 0x0
iInterface 5
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x81    EP 1 IN
    bmAttributes 0x3        INT

```

(下页继续)

```

        wMaxPacketSize 64
        bInterval 8
*** Interface descriptor ***
bLength 9
bDescriptorType 4
bInterfaceNumber 1
bAlternateSetting 0
bNumEndpoints 0
bInterfaceClass 0xa
bInterfaceSubClass 0x0
bInterfaceProtocol 0x0
iInterface 0
*** Interface descriptor ***
bLength 9
bDescriptorType 4
bInterfaceNumber 1
bAlternateSetting 1
bNumEndpoints 2
bInterfaceClass 0xa
bInterfaceSubClass 0x0
bInterfaceProtocol 0x0
iInterface 0
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x82    EP 2 IN
    bmAttributes 0x2        BULK
    wMaxPacketSize 64
    bInterval 0
    *** Endpoint descriptor ***
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x3    EP 3 OUT
    bmAttributes 0x2        BULK
    wMaxPacketSize 64
    bInterval 0

```

速率测试

以太网转 USB 设备 ESP32-S3/P4 连接以太网转 USB 设备 (CH397A)，并开启 softAP，手机连接 softAP 进行网络测速：

芯片	上行 (Mbps)	下行 (Mbps)
ESP32-S3	6.6	7.2
ESP32-P4 with ESP32-C6	11.7	14.7

4G 模组 暂无

资料下载

[CDC1.2 协议文档](#)

API 参考

Header File

- [components/usb/iot_usbh_ecm/include/iot_usbh_ecm.h](#)

Functions

`esp_err_t iot_eth_new_usb_ecm` (const *iot_usbh_ecm_config_t* *config, `iot_eth_driver_t` **ret_handle)

Create a new USB ECM Ethernet driver.

This function initializes a new USB ECM Ethernet driver with the specified configuration. It allocates memory for the driver, sets up the driver functions, and returns a handle to the driver.

参数

- **config** –Pointer to the ECM configuration structure.
- **ret_handle** –Pointer to a location where the handle to the new Ethernet driver will be stored.

返回

- `ESP_OK`: Driver created successfully
- `ESP_ERR_INVALID_ARG`: Invalid argument (NULL config or ret_handle)
- `ESP_ERR_NO_MEM`: Memory allocation failed

Structures

struct `iot_usbh_ecm_config_t`

USB Host Ethernet ECM Configuration.

Public Members

bool **auto_detect**

Auto detect ECM device

TickType_t **auto_detect_timeout**

Auto detect timeout in ticks, used when auto_detect is true

uint16_t **vid**

USB device vendor ID, used when auto_detect is false

uint16_t **pid**

USB device product ID, used when auto_detect is false

int **itf_num**

ecm device notif interface numbers, used when auto_detect is false

5.3 USB 设备驱动

5.3.1 USB Device UVC

`usb_device_uvc` 是用于 ESP32-S2/ESP32-S3/ESP32-P4 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,
};
```

(下页继续)

```
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

`esp_err_t uvc_device_deinit(void)`

Deinitialize the UVC device.

返回 ESP_OK on success ESP_ERR_INVALID_STATE if the UVC device is not initialized, or event group is NULL

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/ESP32-P4 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)
```

5.3.3 ESP TinyUF2

esp_tinyuf2 是一个扩展功能的 [TinyUF2](#)，针对支持 USB-OTG 功能 ESP 芯片开发。支持以下功能：

- 支持通过虚拟 USB 驱动器（U 盘）进行 OTA 升级
- 支持将 NVS 键值输出到虚拟 USB 驱动器中的 ini 文件
- 支持通过修改 ini 文件修改 NVS

UF2 是 Microsoft 开发的文件格式 [PXT](#)，特别适合通过 MSC（存储类）烧录微控制器。有关更友好的解释，请查看 [博客](#)。

在您的项目中支持 UF2 OTA/NVS

1. 使用 `idf.py add_dependency` 命令将组件添加到项目中。

```
idf.py add-dependency "esp_tinyuf2"
```

2. 定义您的分区表。像其他 OTA 解决方案一样，您需要至少保留两个 OTA 应用程序分区。请参考 [Partition Tables](#) 和示例 [usb_uf2_ota](#)。

```
# Partition Table Example
# Name, Type, SubType, Offset, Size, Flags
nvs, data, nvs, , 0x4000,
otadata, data, ota, , 0x2000,
phy_init, data, phy, , 0x1000,
ota_0, app, ota_0, , 1500K,
ota_1, app, ota_1, , 1500K,
```

3. 使用 `idf.py menuconfig` 配置组件 (Top) → Component config → TinyUF2 Config
 - USB Virtual Disk size (MB): 虚拟 USB 驱动器在文件资源管理器中显示的大小，默认情况下为 8MB
 - Max APP size (MB): 最大应用大小，默认情况下 4MB
 - Flash cache size (KB): 缓存大小，用于缓存即将烧录的 bin 片段，默认情况下为 32KB
 - USB Device VID: Espressif VID (默认 0x303A)
 - USB Device PID: Espressif test PID (默认 0x8000)，如需申请 PID 请参考 [esp-usb-pid](#)。
 - USB Disk Name: 虚拟 USB 驱动器在文件资源管理器中显示的名称，ESP32Sx-UF2 默认
 - USB Device Manufacture: 制造商 Espressif 默认
 - Product Name: 产品名称 ESP TinyUF2 默认
 - Product ID: 产品 ID 12345678 默认
 - Product URL: index 文件将被添加到 U 盘，用户可以单击进入网页，<https://products.espressif.com/> 默认
 - UF2 NVS ini file size: ini 文件大小，用于保存 NVS 键值对
4. 如下所示，安装 tinyuf2 功能，有关更多详细信息，请参阅示例 [usb_uf2_nvs](#) 和 [usb_uf2_ota](#)

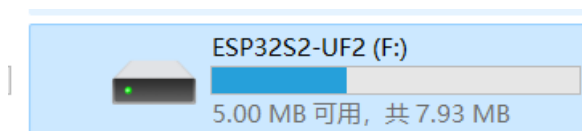
```
/* install UF2 OTA */
tinyuf2_ota_config_t ota_config = DEFAULT_TINYUF2_OTA_CONFIG();
ota_config.complete_cb = uf2_update_complete_cb;
/* disable auto restart, if false manual restart later */
ota_config.if_restart = false;
/* install UF2 NVS */
tinyuf2_nvs_config_t nvs_config = DEFAULT_TINYUF2_NVS_CONFIG();
```

(下页继续)

(续上页)

```
nvs_config.part_name = "nvs";
nvs_config.namespace_name = "myuf2";
nvs_config.modified_cb = uf2_nvs_modified_cb;
esp_tinyuf2_install(&ota_config, &nvs_config);
```

5. 运行 `idf.py build flash` 进行初始下载, 之后可以使用 `idf.py uf2-ota` 生成新的 uf2 app bin
6. 将 UF2 格式 app bin 拖放到磁盘, 升级将自动进行



UF2 NVS 隐藏

通过 `menuconfig` (Top) → Component config → TinyUF2 Config → UF2 ini file hide NVS value, NVS 的值将会被加密并替换为 ****.

```
esp_tinyuf2_add_key_hidden("password")
```

使用 UF2 USB Console

通过 `menuconfig` (Top) → Component config → TinyUF2 Config → Enable USB Console For log, 该日志将输出到 USB 串行端口 (默认向 UART 输出) .

编译 APP 为 UF2 格式

该组件添加了新的命令 `idf.py uf2-ota`, 可以用于将 app 编译为 UF2 格式. 编译完成后, 新的 UF2 文件 (`${PROJECT_NAME}.uf2`) 将被产生到 project 文件夹.

```
idf.py uf2-ota
```

将现有 bin 转换为 UF2 格式

将您现有的应用 bin 转换为 UF2 格式, 可以使用 `uf2conv.py`, 您需要指定 family id 为 ESP32S2, ESP32S3 或如下的 magic number. 并且还需使用 `-b` 指定地址, `tinyuf2` 将使用它作为偏移来写入 OTA 分区.

1. 转换指令
using:

```
uf2conv.py your_firmware.bin -c -b 0x00 -f ESP32S3
```

or:

```
uf2conv.py your_firmware.bin -c -b 0x00 -f 0xc47e5767
```

注意

- 要连续使用 UF2 OTA 功能, 必须在更新的应用中同样启用 `tinyuf2`.

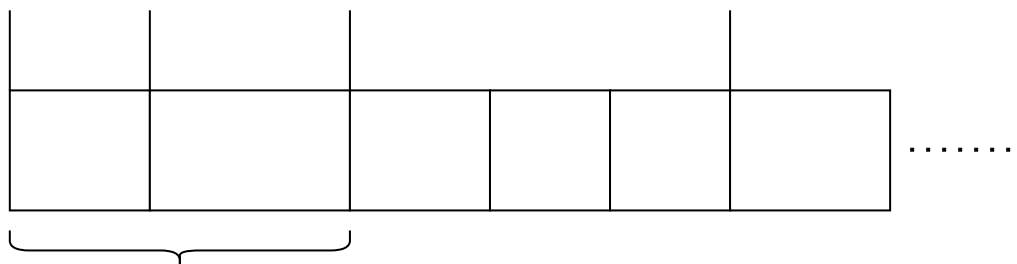
在 Bootloader 中使用 UF2

通过将带有 UF2 功能的特殊 APP Bin 隐藏到 Bootloader 中，可以做到以下功能 1. 当 factory/test/ota 分区无固件时自动进入 UF2 下载模式 2. 手动拉低 BOOT_UF2 引脚自动进入 UF2 下载模式 3. 在用户 APP 中手动调用 `esp_restart_from_tinyuf2()` 函数进入 UF2 下载模式

示例 `usb/device/bootloader_uf2`

使用说明：

1. 默认仅支持 nvs/phy_init/factory 分区，如需支持 test/ota/spiffs 分区，请手动修改分区表，并重新编译。
2. 拖拽升级默认升级 factory 分区，请确保存在 factory 分区或手动修改代码。
3. 默认 CONFIG_PARTITION_TABLE_OFFSET 大小为 0x60000，如固件过大，请修改该值
4. 默认选用的 nvs 分区名称为 CONFIG_BOOTLOADER_UF2_NVS_PART_NAME (“nvs”)，默认的 nvs 命名空间为 CONFIG_BOOTLOADER_UF2_NVS_NAMESPACE_NAME (“uf2_nvs”)，请确保用户固件中使用的 NVS 分区名称和命名空间与 bootloader 中一致才支持修改。
5. 默认支持工作指示灯闪烁，指示灯默认为 CONFIG_BOOTLOADER_UF2_LED_INDICATOR_GPIO_NUM (2)，请确保与硬件连接保持一致。



Flash 分区参考：

注意事项

- Bootloader UF2 bin 应该烧录在 CONFIG_MMU_PAGE_SIZE 对齐的整数倍地址上，否则将无法正常工作。

Note: ESP32-S2/ESP32-S3/ESP32-P4 的 CONFIG_MMU_PAGE_SIZE 默认为 64KB (0x10000)，因此需要烧录在 64KB (0x10000) 对齐的地址上。

- 需要将 CONFIG_PARTITION_TABLE_OFFSET 配置大小改为 bootloader.bin + bootloader_uf2.bin 的大小，否则无法烧录进去 bootloader_uf2.bin。
- 需要将 CONFIG_ENABLE_BOOTLOADER_UF2 宏打开
- 需要将 CONFIG_SPI_FLASH_DANGEROUS_WRITE_ALLOWED 宏打开：因为 bootloader_uf2.bin 烧录位置并未显示在 partition-tables 中，所以需要关闭检查。
- 需要将 Bin 文件按照以下方式地址烧录进固件：

1. CONFIG_BOOTLOADER_OFFSET_IN_FLASH(The starting addresses vary for different chips.) - bootloader.bin
2. 0x10000 - bootloader_uf2.bin
3. CONFIG_PARTITION_TABLE_OFFSET(0x60000) - partition-table.bin

可以将下列片段添加到工程的 CMakeLists.txt 中，会在编译结束后自行合成 Bin 文件：

```
add_custom_command(
  TARGET app
  POST_BUILD
  COMMAND ${CMAKE_COMMAND} -E echo "Flash merged bin merge_uf2.bin to address $
  ↪ ${CONFIG_BOOTLOADER_OFFSET_IN_FLASH}"
```

(下页继续)

(续上页)

```

COMMAND ${ESPTOOLPY} --chip ${IDF_TARGET} merge_bin -o merge_uf2.bin ${CONFIG_
↪BOOTLOADER_OFFSET_IN_FLASH} ${BUILD_DIR}/bootloader/bootloader.bin 0x10000 $
↪${BUILD_DIR}/${PROJECT_BIN} ${CONFIG_PARTITION_TABLE_OFFSET} ${BUILD_DIR}/
↪partition_table/partition-table.bin
)

```

API 参考

Header File

- [components/usb/esp_tinyuf2/esp_tinyuf2.h](#)

Functions

`esp_err_t esp_tinyuf2_install (tinyuf2_ota_config_t *ota_config, tinyuf2_nvs_config_t *nvs_config)`

Flashing app to specified partition through USB UF2 (Virtual USB Disk), and support operate NVS partition through USB UF2 CONFIG.ini file.

参数

- **ota_config** –tinyuf2 configs described in *tinyuf2_ota_config_t*
- **nvs_config** –tinyuf2 nvs configs described in *tinyuf2_nvs_config_t*

返回

- ESP_ERR_INVALID_ARG invalid parameter, please check partitions
- ESP_ERR_INVALID_STATE tinyuf2 already installed
- ESP_OK Success

`esp_err_t esp_tinyuf2_uninstall (void)`

Uninstall tinyuf2, only reset USB to default state.

备注: not release memory due to tinyusb not support teardown

返回 `esp_err_t`

- ESP_ERR_INVALID_STATE tinyuf2 not installed
- ESP_OK Success

`tinyuf2_state_t esp_tinyuf2_current_state (void)`

Get tinyuf2 current state.

返回 `tinyuf2_state_t`

`void esp_restart_from_tinyuf2 (void)`

Restart system and set reset reason to UF2_RESET_REASON_VALUE.

Structures

struct **tinyuf2_ota_config_t**

tinyuf2 configurations

Public Members

`esp_partition_subtype_t subtype`

Partition subtype. if ESP_PARTITION_SUBTYPE_ANY will use the next_update_partition by default.

const char ***label**

Partition label. Set this value if looking for partition with a specific name. if subtype==ESP_PARTITION_SUBTYPE_ANY, label default to NULL.

bool **if_restart**

if restart system to new app partition after UF2 flashing done

update_complete_cb_t **complete_cb**

user callback called after uf2 update complete

struct **tinyuf2_nvs_config_t**

tinyuf2 nvs configurations

Public Members

const char ***part_name**

Partition name.

const char ***namespace_name**

Namespace name.

nvs_modified_cb_t **modified_cb**

user callback called after uf2 update complete

Macros

DEFAULT_TINYUF2_OTA_CONFIG()

DEFAULT_TINYUF2_NVS_CONFIG()

UF2_RESET_REASON_VALUE

Type Definitions

typedef void (***update_complete_cb_t**)(void)

user callback called after uf2 update complete

typedef void (***nvs_modified_cb_t**)(void)

user callback called after nvs modified

Enumerations

enum **tinyuf2_state_t**

tinyuf2 current state

Values:

enumerator **TINYUF2_STATE_NOT_INSTALLED**

tinyuf2 driver not installed

enumerator **TINYUF2_STATE_INSTALLED**

tinyuf2 driver installed

enumerator **TINYUF2_STATE_MOUNTED**

USB mounted

Chapter 6

音频

6.1 PWM 音频

PWM 音频功能使用 ESP32 系列芯片的内部 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

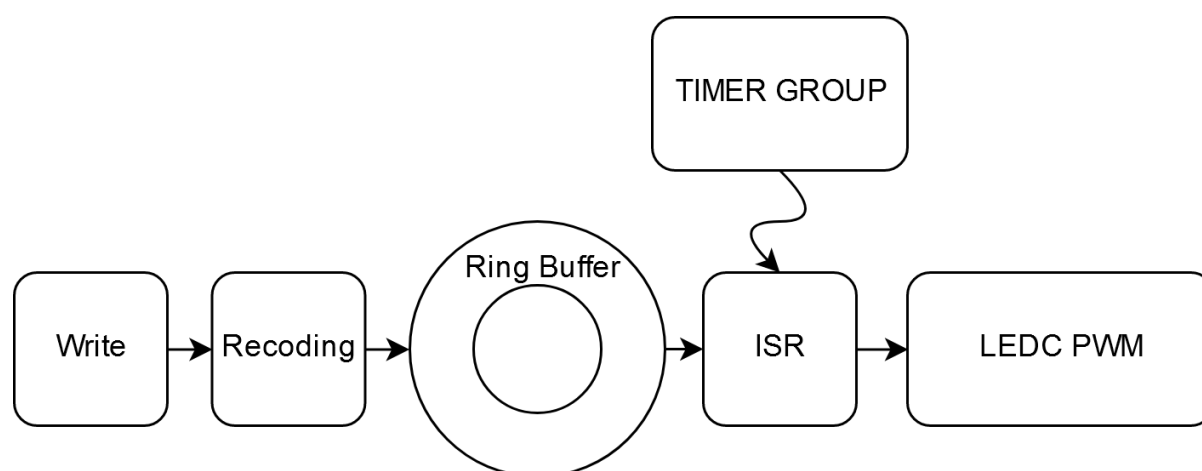


图 1: 结构

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

6.3 ADC 麦克风

使用 ADC 采集模拟麦克风数据，无需使用外部的音频 codec 芯片，适用于对采样精度要求不太高且成本敏感的应用。在代码结构上采用 `esp_codec_dev` 的工程模式。

6.3.1 特性

- 支持多路 ADC 音频采样。
- 支持最大 12 位的 ADC 采样分辨率。

备注：多路采样支持的最大分辨率为，ADC 最大采样率 (S3 为 83.3K) / 采样通道数。

6.3.2 参考电路

外部参考电路如下，可以选择跨阻放大器或者同相/反相放大电路对 MIC 输入信号进行放大。在大多数情况下，使用跨阻放大器可以取得更佳的信噪比，在本示例中 R29 用于为 MIC 添加直流偏置，使用通用运放 LMV321 作为运算放大器，通过 R26 与 C19 构成反馈电路，使运放工作在跨阻放大模式，其中阻容根据 MIC 的灵敏度进行调整。R31 与 R30 用于设定中位电压，以避免交越失真，R27 与 R32 用于提供直流偏置，避免 ADC 采集时失真。关于前端 MIC 放大器的更多设计参考信息，请参考 [Single-Supply, Electret Microphone Pre-Amplifier Reference Design](#)。

备注：

- 偏置：ADC 端的中点偏置是必须的（如 $V_{ref}/2$ ），避免 ADC 采集失真，默认情况下 ESP32-C3 的 V_{ref} 约为 0.9V。
 - 电源：MIC 供电最好使用 LDO 进行稳压，避免电源波动影响 ADC 采样。如果不使用外部 LDO，请在电源侧进行单独的 RC 滤波（在本示例中 R28 与 C23 用于滤波），MIC 推荐与运放独立供电。
 - 推荐选择灵敏度大于 -46dB 的 MIC，避免放大后的信噪比过低。并根据 MIC 的灵敏度调整放大倍数。
-

6.3.3 外部配置 ADC Continuous

可在外部初始化好 adc continuous mode, 并将 `adc_continuous_handle_t` 传入。

MIC

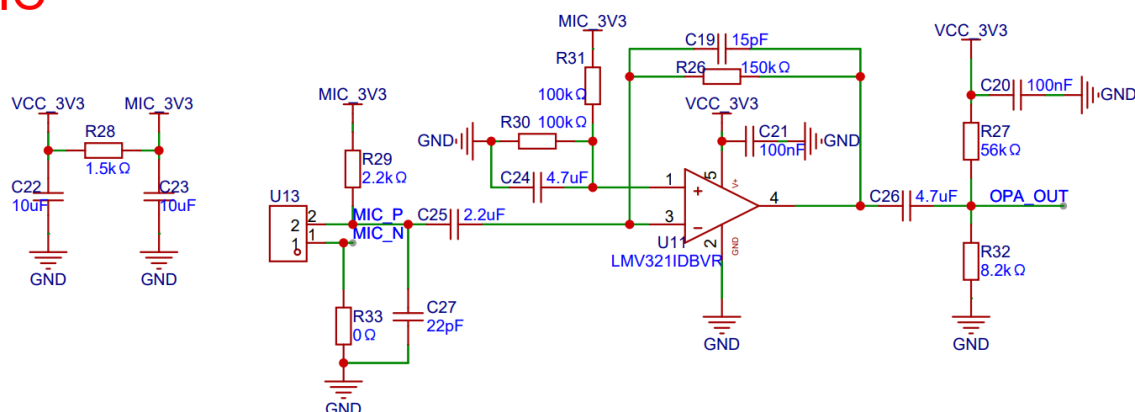


图 2: ADC mic hardware reference design

6.3.4 与 ADC oneshot 模式一起使用

- 如果 ADC oneshot 与 ADC condition 使用不同的 ADC Unit, 那么不会有冲突。
- 如果使用相同的 ADC Unit, 请调用 `esp_codec_dev_close()` 关闭 `adc_mic` 时调用 `adc_oneshot_read`。

```
adc_continuous_start(handle);
adc_continuous_read();
adc_continuous_stop();
adc_oneshot_read();
```

6.3.5 示例代码

```
audio_codec_adc_cfg_t cfg = DEFAULT_AUDIO_CODEC_ADC_MONO_CFG(ADC_CHANNEL_0, 16000);
const audio_codec_data_if_t *adc_if = audio_codec_new_adc_data(&cfg);

esp_codec_dev_cfg_t codec_dev_cfg = {
    .dev_type = ESP_CODEC_DEV_TYPE_IN,
    .data_if = adc_if,
};
esp_codec_dev_handle_t dev = esp_codec_dev_new(&codec_dev_cfg);
esp_codec_dev_sample_info_t fs = {
    .sample_rate = 16000,
    .channel = 1,
    .bits_per_sample = 16,
};
esp_codec_dev_open(dev, &fs);
uint16_t *audio_buffer = malloc(16000 * sizeof(uint16_t));
assert(audio_buffer);
while (1) {
    int ret = esp_codec_dev_read(dev, audio_buffer, sizeof(uint16_t) * 16000);
    ESP_LOGI(TAG, "esp_codec_dev_read ret: %d\n", ret);
}
```

6.3.6 API 参考

Header File

- `components/audio/adc_mic/adc_mic.h`

Functions

`const audio_codec_data_if_t *audio_codec_new_adc_data (audio_codec_adc_cfg_t *adc_cfg)`

Initialize codec adc.

参数 **adc_cfg** –pointer of configuration struct

返回 `const audio_codec_data_if_t*` adc data interface.

Structures

struct **audio_codec_adc_cfg_t**

codec adc configuration

Public Members

`adc_continuous_handle_t *handle`

Type of adc continuous mode driver handle, if NULL will create new one internal, else will use the handle

`size_t max_store_buf_size`

Max length of the conversion results that driver can store, in bytes.

`size_t conv_frame_size`

Conversion frame size, in bytes. This should be in multiples of `SOC_ADC_DIGI_DATA_BYTES_PER_CONV`

`adc_unit_t unit_id`

ADC unit

`uint8_t *adc_channel_list`

Channel of ADC

`uint8_t adc_channel_num`

Number of channels

`adc_atten_t atten`

It should correspond to the actual range, with 0dB attenuation having the least ripple.

`uint32_t sample_rate_hz`

Sample rate of ADC

Macros

DEFAULT_AUDIO_CODEC_ADC_MONO_CFG (`_channel`, `_sample`)

DEFAULT_AUDIO_CODEC_ADC_STEREO_CFG (`_channel_l`, `_channel_r`, `_sample`)

Chapter 7

音视频编解码

7.1 AVI 播放器

该组件支持解析 AVI 文件，并按照 FPS 速率解析 AVI 文件，将音视频数据已回调的方式提供给用户。

AVI (Audio Video Interleave) 文件是一种多媒体容器格式，用于存储音频和视频数据。它可以包含多种编码格式的视频和音频流，并且支持不同的压缩方式。

该组件目前仅支持以下格式的音视频

Video:

- MJPEG
- H264

Audio:

- PCM

7.1.1 API 参考

Header File

- [components/avi_player/include/avi_player.h](#)

Functions

`esp_err_t avi_player_play_from_memory` ([avi_player_handle_t](#) handle, `uint8_t *avi_data`, `size_t avi_size`)

Plays an AVI file from memory. The buffer of the AVI will be passed through the set callback function.

This function initializes and plays an AVI file from a memory buffer.

参数

- **handle** –[in] AVI player handle
- **avi_data** –[in] Pointer to the AVI file data in memory.
- **avi_size** –[in] Size of the AVI file data in bytes.

返回 `esp_err_t` ESP_OK if successful, otherwise an error code.

esp_err_t **avi_player_play_from_file** (*avi_player_handle_t* handle, const char *filename)

Plays an AVI file from the filesystem. The buffer of the AVI will be passed through the set callback function.

This function initializes and plays an AVI file from the filesystem using its filename.

参数

- **handle** –[in] AVI player handle
- **filename** –[in] Path to the AVI file on the filesystem.

返回 esp_err_t ESP_OK if successful, otherwise an error code.

esp_err_t **avi_player_get_video_buffer** (*avi_player_handle_t* handle, void **buffer, size_t *buffer_size, *video_frame_info_t* *info, TickType_t ticks_to_wait)

Get one video frame from AVI stream.

参数

- **handle** –[in] AVI player handle
- **buffer** –[out] Pointer to external buffer to hold one frame
- **buffer_size** –[inout] Size of external buffer
- **info** –[out] Information of the video frame
- **ticks_to_wait** –[in] Maximum blocking time

返回

- ESP_OK Success
- ESP_ERR_TIMEOUT Timeout
- ESP_ERR_INVALID_ARG NULL arguments
- ESP_ERR_NO_MEM External buffer not enough

esp_err_t **avi_player_get_audio_buffer** (*avi_player_handle_t* handle, void **buffer, size_t *buffer_size, *audio_frame_info_t* *info, TickType_t ticks_to_wait)

Get the audio buffer from AVI file.

参数

- **handle** –[in] AVI player handle
- **buffer** –[out] pointer to the audio buffer
- **buffer_size** –[in] size of the audio buffer
- **info** –[out] audio frame information
- **ticks_to_wait** –[in] maximum blocking time in ticks

返回

- ESP_OK on success
- ESP_ERR_TIMEOUT if semaphore is not acquired before timeout
- ESP_ERR_INVALID_ARG if buffer or info is NULL or buffer_size is zero
- ESP_ERR_NO_MEM if buffer size is not enough

esp_err_t **avi_player_play_stop** (*avi_player_handle_t* handle)

Stop AVI player.

参数 **handle** –[in] AVI player handle

返回

- ESP_OK: Stop AVI player successfully
- ESP_ERR_INVALID_STATE: AVI player not playing

esp_err_t **avi_player_init** (*avi_player_config_t* config, *avi_player_handle_t* *handle)

Initialize the AVI player.

参数

- **config** –[in] Configuration of AVI player
- **handle** –[out] Pointer to store the AVI player handle

返回

- ESP_OK: succeed
- ESP_ERR_NO_MEM: Cannot allocate memory for AVI player
- ESP_ERR_INVALID_STATE: AVI player has already been initialized

esp_err_t **avi_player_deinit** (*avi_player_handle_t* handle)

Deinitializes the AVI player.

This function deinitializes and cleans up resources used by the AVI player.

参数 **handle** –[in] AVI player handle

返回 esp_err_t ESP_OK if successful, otherwise an error code.

Structures

struct **video_frame_info_t**

video frame info

Public Members

uint32_t **width**

Width of image in pixels

uint32_t **height**

Height of image in pixels

video_frame_format **frame_format**

Pixel data format

struct **audio_frame_info_t**

audio frame info

Public Members

uint8_t **channel**

Audio output channel

uint8_t **bits_per_sample**

Audio bits per sample

uint32_t **sample_rate**

Audio sample rate

audio_frame_format **format**

Audio format

struct **frame_data_t**

frame data

Public Members

uint8_t ***data**

Image data for this frame

size_t **data_bytes**

Size of image data buffer

frame_type_t **type**

Frame type: video or audio

video_frame_info_t **video_info**

Video frame info

audio_frame_info_t **audio_info**

Audio frame info

union *frame_data_t*::[anonymous] [**anonymous**]

frame info

struct **avi_player_config_t**

avi player config

Public Members

size_t **buffer_size**

Internal buffer size

video_write_cb **video_cb**

Video frame callback

audio_write_cb **audio_cb**

Audio frame callback

audio_set_clock_cb **audio_set_clock_cb**

Audio set clock callback

avi_play_end_cb **avi_play_end_cb**

AVI play end callback

UBaseType_t **priority**

FreeRTOS task priority

BaseType_t **coreID**

ESP32 core ID

void ***user_data**

User data

int **stack_size**

Stack size for the player task

bool **stack_in_psram**

If you read file/data from flash, do not set true

Type Definitions

```
typedef void (*video_write_cb)(frame_data_t *data, void *arg)
```

```
typedef void (*audio_write_cb)(frame_data_t *data, void *arg)
```

```
typedef void (*audio_set_clock_cb)(uint32_t rate, uint32_t bits_cfg, uint32_t ch, void *arg)
```

```
typedef void (*avi_play_end_cb)(void *arg)
```

```
typedef void *avi_player_handle_t
```

Enumerations

```
enum video_frame_format
```

video frame format

Values:

enumerator **FORMAT_MJPEG**

enumerator **FORMAT_H264**

```
enum audio_frame_format
```

audio frame format

Values:

enumerator **FORMAT_PCM**

```
enum frame_type_t
```

frame type: video or audio

Values:

enumerator **FRAME_TYPE_VIDEO**

enumerator **FRAME_TYPE_AUDIO**

Chapter 8

图形界面

8.1 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。

8.1.1 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

8.1.2 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

8.1.3 在线工具

LVGL 提供了在线的 [字模提取工具](#) 和 [图片取模工具](#)。

8.1.4 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](#)

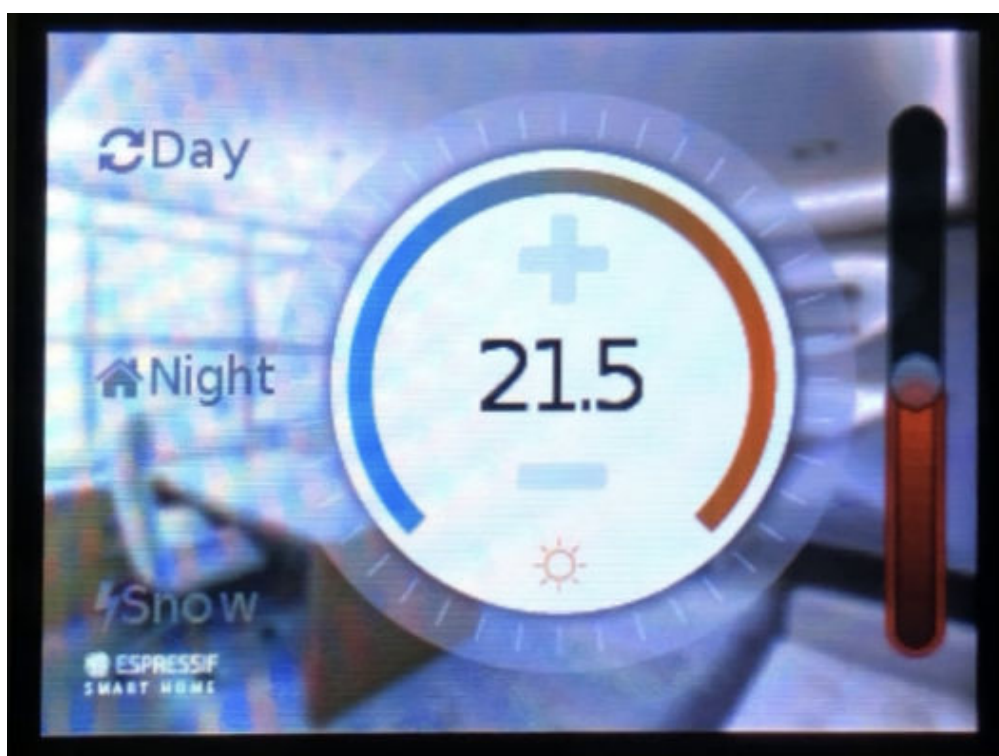
官方例程

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

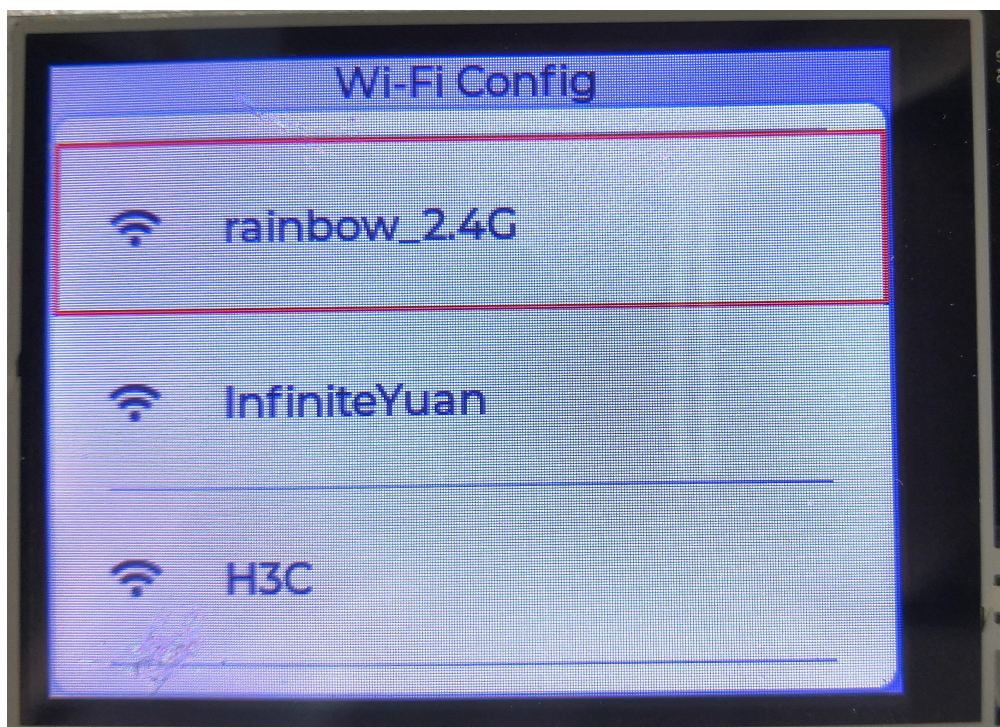
使用 LVGL 绘制了一个咖啡机的交互界面：

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 9

人工智能

9.1 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。

该库支持以下 OpenAI API 能力：

- ☐ [Responses](#)¹
- ☒ [Chat Completions](#): 支持 text, image 与 audio 输入，输出支持 text。² 暂不支持流式输出。
- ☒ [Audio](#): Speech & Transcription，其中 Speech 支持流式输出。
- ☒ [Images](#): Create image & Edits & Variations
- ☒ [Embeddings](#)
- ☐ [Evals](#)
- ☐ [Fine-tuning](#)
- ☐ [Batch](#)
- ☐ [Files](#)
- ☒ [Moderations](#)

备注：Realtime 能力可以使用以下库实现：

- **WebSocket 方案**: - <https://github.com/openai/openai-realtime-embedded>
 - **WebRTC 方案**: - https://github.com/espressif/esp-webrtc-solution/tree/main/solutions/openai_demo
-

更多详情，请参考 [OpenAI](#) 的 API 参考文档。

9.1.1 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。

¹ 目前，请使用 Chat Completions API 代替。

² 多模态输入需要使用 `multiModalMessage`。

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。

9.1.2 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**

OpenAI_StringResponse_t *(***multiModalMessage**)(struct *OpenAI_ChatCompletion* *chatCompletion, const char *type, const char *p, bool save)

Send a multi-modal message (text / image_url / input_audio) for completion. This is an extended API supporting richer content types.

Param chatCompletion[in] The chat completion object

Param type[in] The content type string ("text" , "image_url" , "input_audio" , etc.)

Param p[in] The content value corresponding to the type

Param save[in] Save conversation with the first assistant response if true

Return *OpenAI_StringResponse_t** The response from *OpenAI*

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.

Param createSpeech[in] the point of *OpenAI_SpeechResponse_t*

Param p[in] the message for audio generation

Return *

void (***speechStream**)(struct *OpenAI_AudioSpeech* *createSpeech, char *p, *OpenAI_StreamCallback* stream_callback)

Send the message for completion.

Param createSpeech[in] the point of OpenAI_SpeechResponse_t
Param p[in] the message for audio generation
Param stream_callback[in] the callback function for audio stream

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 void (***OpenAI_StreamCallback**)(const uint8_t *data, size_t length)

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
```

```
enumerator OPENAI_AUDIO_OUTPUT_FORMAT_WAV
```

```
enumerator OPENAI_AUDIO_OUTPUT_FORMAT_PCM
```

```
enumerator OPENAI_AUDIO_OUTPUT_FORMAT_MAX
```

9.2 Touch 手写数字识别

9.2.1 Touch 原理与数据采集

Touch 原理

采用 [ESP_Touch_Kit_Touchpad](#) 作为 Touch 触摸板。该触摸板有 6*7 的 Touch 通道组成。

当手指在 Touch 触摸板上移动时，会改变 Touch 通道的电容值，这时候我们就可以通过检测电容值的变化来判断手指的位置。

检测算法

1. 因为硬件差异，各个通道触发的最大值和最小值不同，所以需要各个通道的电容值进行归一化处理。通过手指在上面滑动，来记录各个通道的最大值和最小值。
2. 在手指滑动的过程中，会有一路通道的电容值变化率最大，这时候再取相邻变化率最大的通道，可以获得到两条相邻的通道。
3. 将两个通道的值进行差比和运算，可以获取到手指在该方向上的相对坐标（较两通道中心点的偏移量）。

$$x = \frac{Fa - Fb}{Fa + Fb}$$

4. 通过上述步骤，可以获取到手指在两个方向上的相对坐标，从而可以确定手指的位置。

备注：需要设定合适的触发阈值，来判断手指是否在 Touch 触摸板上移动。

5. 当检测到手指抬起，即可将绘制的数据进行保存。

数据采集

由于在实际的 Touch 触控板绘制数字时，其最终的图像并非类似于 MNIST 数据集中的手写数字，导致直接使用 MNIST 数据集训练的模型在实际应用中效果不佳。因此，需要采集实际的 Touch 触控板绘制数字的数据集，并进行训练。

备注：经过插值后的手写数据图像尺寸为 30x25

点击此处下载本示例中使用的数据集：[touch_dataset.zip](#)

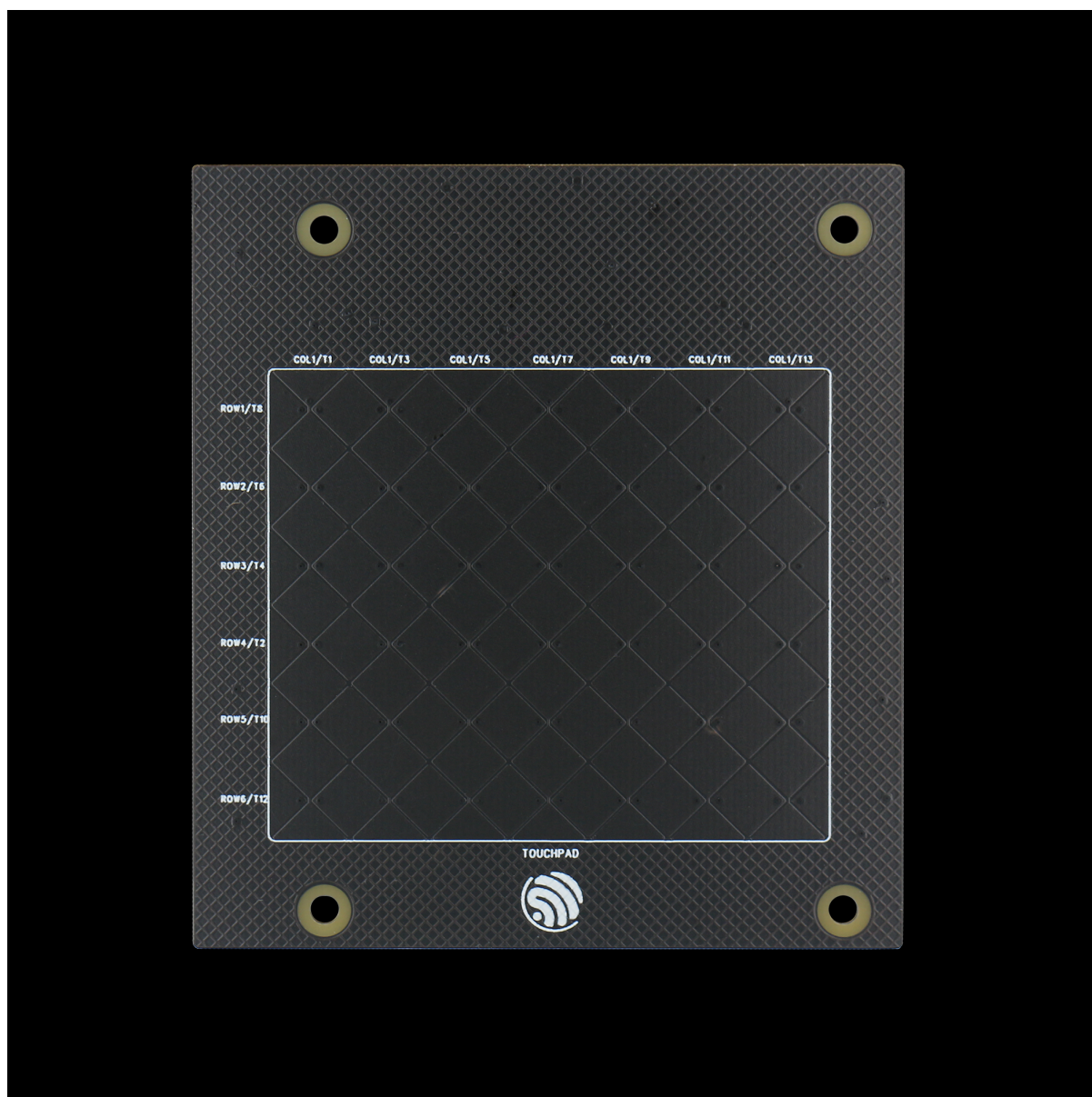


图 1: Touch 触控板实物图

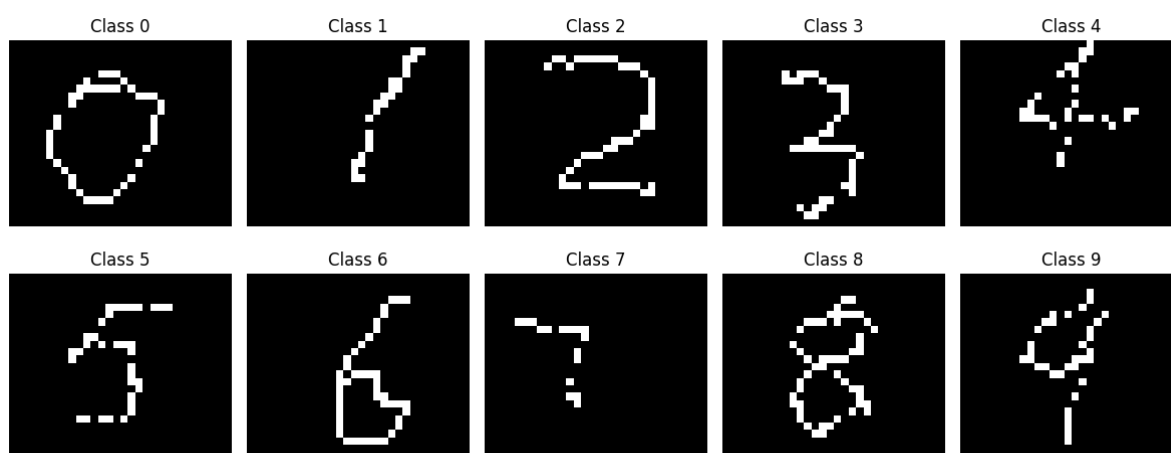


图 2: 基于 Touch 触控板绘制的真实数据集

9.2.2 模型训练与部署

模型搭建

基于 PyTorch 框架，搭建适用于 Touch 手写数字的网络模型，结构如下：

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.model = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
↪stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),

            torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
↪stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),

            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,
↪stride=1, padding=1),
            torch.nn.ReLU(),

            torch.nn.Flatten(),
            torch.nn.Linear(in_features=7 * 6 * 64, out_features=256),
            torch.nn.ReLU(),
            torch.nn.Dropout(p=0.5),
            torch.nn.Linear(in_features=256, out_features=10),
            torch.nn.Softmax(dim=1)
        )

    def forward(self, x):
        output = self.model(x)
        return output
```

模型训练

模型的训练过程包括数据集的加载与预处理，模型训练参数的设置，模型训练过程的监控与保存。

数据加载与预处理 将不同数字对应的图像分别放置在 dataset/extra 文件夹中，以数字作为文件夹名称，并使用 transforms.Compose 对图像进行预处理，包括灰度化、随机旋转平移与标准化。随后使用 ImageFolder 加载整个数据集，并按 8:2 的比例划分为训练集与测试集，最后通过 DataLoader 构建批量加载器以供后续模型训练和评估使用。

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms

transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.RandomAffine(degrees=10, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
])
```

(下页继续)

(续上页)

```
dataset = datasets.ImageFolder(root='./dataset/extra', transform=transform)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=False)
```

模型训练参数设定 模型训练参数包括学习率、优化器、损失函数等。实际训练过程中采用交叉熵作为损失函数，并使用 Adam 优化器进行模型参数的更新：

```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
model = Net().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

模型训练与保存 设置训练轮数为 100 轮，并使用训练集和测试集分别进行训练和评估。训练过程中使用训练集进行模型参数的更新，并使用测试集进行模型性能的评估。训练结束后，将模型参数保存为 ./models/final_model.pth 文件。

```
def train_epoch(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_acc = 100 * correct / total
    return epoch_loss, epoch_acc

def test_epoch(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
```

(下页继续)

(续上页)

```

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(test_loader)
    epoch_acc = 100 * correct / total
    return epoch_loss, epoch_acc

num_epochs = 100
train_acc_array = []
test_acc_array = []
for epoch in range(num_epochs):
    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, device)
    test_loss, test_acc = test_epoch(model, test_loader, criterion, device)

    print(f'Epoch [{epoch + 1}/{num_epochs}], '
          f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, '
          f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%')
    train_acc_array.append(train_acc)
    test_acc_array.append(test_acc)

torch.save(model.state_dict(), './models/final_model.pth')

```

模型训练过程中，训练集与测试集的准确率变化曲线如下：

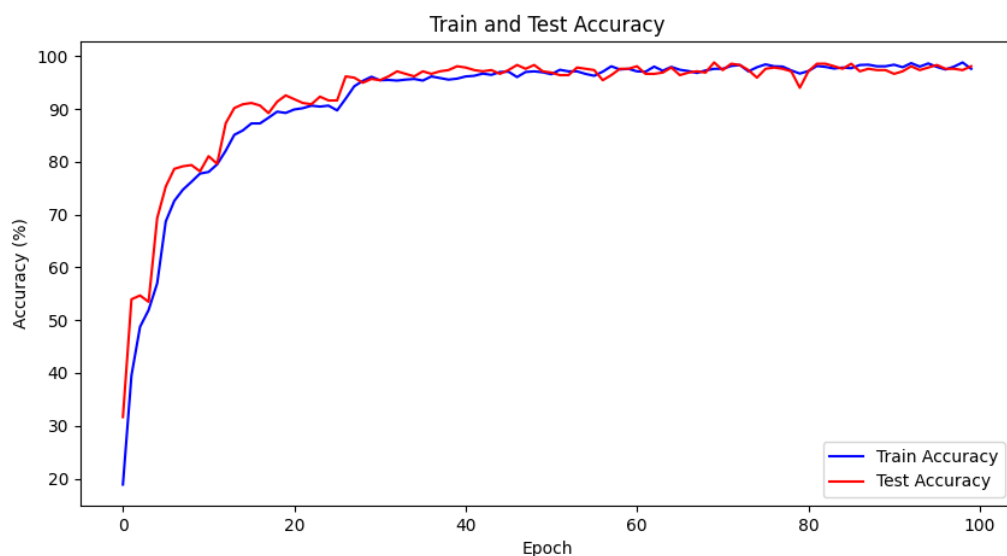


图 3: 训练集与测试集的准确率变化曲线

模型部署

ESP-PPQ 环境配置 ESP-PPQ 是一种基于 ppq 的量化工具。请使用以下命令安装 ESP-PPQ：

```

pip uninstall ppq
pip install git+https://github.com/espressif/esp-ppq.git

```

模型量化与部署 参考 [How to quantize model](#) 实现对模型量化与导出。若需要导出适用于 ESP32P4 的模型，请将 TARGET 设置为 esp32p4。

```
import torch
from PIL import Image
from ppq.api import espdl_quantize_torch
from torch.utils.data import Dataset
from torch.utils.data import random_split
from torchvision import transforms, datasets

DEVICE = "cpu"

class FeatureOnlyDataset(Dataset):
    def __init__(self, original_dataset):
        self.features = []
        for item in original_dataset:
            self.features.append(item[0])

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        return self.features[idx]

def collate_fn2(batch):
    features = torch.stack(batch)
    return features.to(DEVICE)

if __name__ == '__main__':
    BATCH_SIZE = 32
    INPUT_SHAPE = [1, 25, 30]
    TARGET = "esp32s3"
    NUM_OF_BITS = 8
    ESPDL_MODEL_PATH = "./s3/touch_recognition.espdl"

    transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize((0.5, ), (0.5, )),
    ])

    dataset = datasets.ImageFolder(root="../dataset/extra", transform=transform)
    train_size = int(0.8 * len(dataset))
    test_size = len(dataset) - train_size
    train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

    image = Image.open("../dataset/extra/9/20250225_140331.png").convert('L')
    input_tensor = transform(image).unsqueeze(0)
    print(input_tensor)

    feature_only_test_data = FeatureOnlyDataset(test_dataset)

    testDataLoader = torch.utils.data.DataLoader(dataset=feature_only_test_data,
    ↪batch_size=BATCH_SIZE, shuffle=False,
                                         collate_fn=collate_fn2)

    model = Net().to(DEVICE)
    model.load_state_dict(torch.load("../final_model.pth", map_location=DEVICE))
    model.eval()
```

(下页继续)

(续上页)

```
quant_ppq_graph = espdL_quantize_torch(
    model=model,
    espdL_export_file=ESPDL_MODEL_PATH,
    calib_dataloader=testDataLoader,
    calib_steps=8,
    input_shape=[1] + INPUT_SHAPE,
    inputs=[input_tensor],
    target=TARGET,
    num_of_bits=NUM_OF_BITS,
    device=DEVICE,
    error_report=True,
    skip_export=False,
    export_test_values=True,
    verbose=1,
    dispatching_override=None
)
```

为了便于调试模型，ESP-DL 提供了在量化期间添加测试数据并在 PC 端查看推理结果的功能。在上述过程中，image 被加载至 espdL_quantize_torch 中被用于测试。在模型转换结束后，测试数据的推理结果将在以 *.info 为后缀的文件中保存：

```
test outputs value:
%23, shape: [1, 10], exponents: [0],
value: array([9.85415445e-34, 1.92874989e-22, 7.46892081e-43, 1.60381094e-28,
              3.22134028e-27, 1.05306175e-20, 4.07960022e-41, 1.42516404e-21,
              2.38026637e-26, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
            dtype=float32)
```

重要：在模型量化与部署过程中，请将 torch.utils.data.DataLoader 中的 shuffle 参数设置为 False。

9.2.3 端侧推理

参考 [How to load test profile model](#) 与 [How to run model](#) 实现模型的加载与推理。

需要注意的是，在本例中，Touch 驱动上报的按下与未按下的状态为 1 与 0，而模型的输入为标准化后的图像数据，因此需要对 Touch 驱动上报的数据进行预处理：

```
for (size_t i = 0; i < m_feature_size; i++) {
    int8_t value = (input_data[i] == 0 ? -1 : 1);
    quant_buffer[i] = dl::quantize<int8_t>((float)value, m_input_scale);
}
```

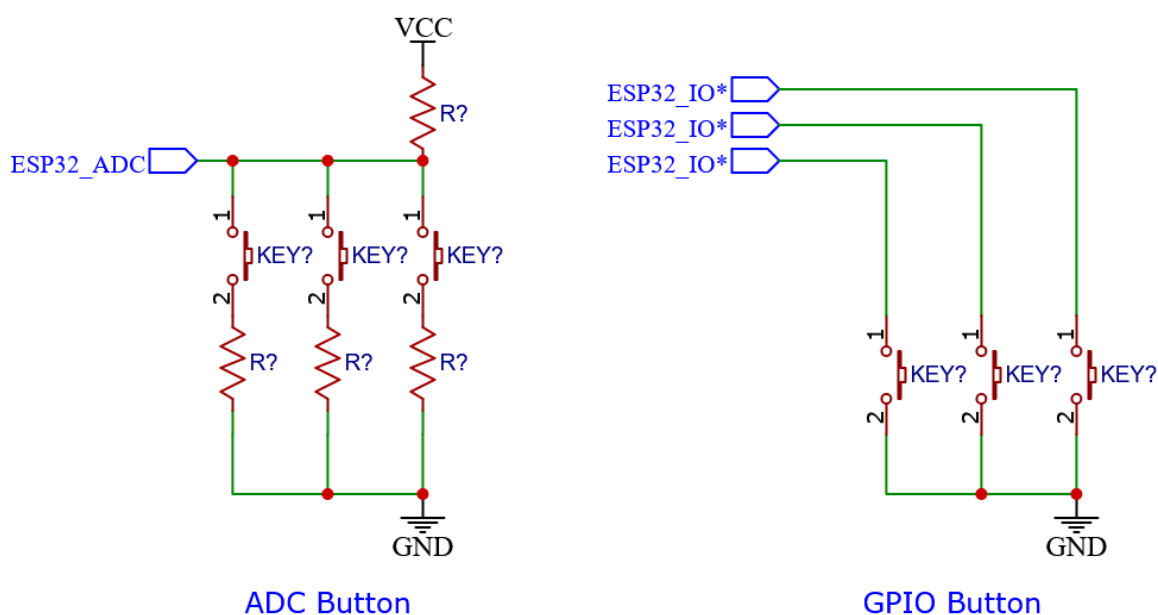
完整的项目工程请参考：[ai/esp_dl/touchpad_digit_recognition](#)

Chapter 10

输入设备

10.1 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。

10.1.1 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 ≥ 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

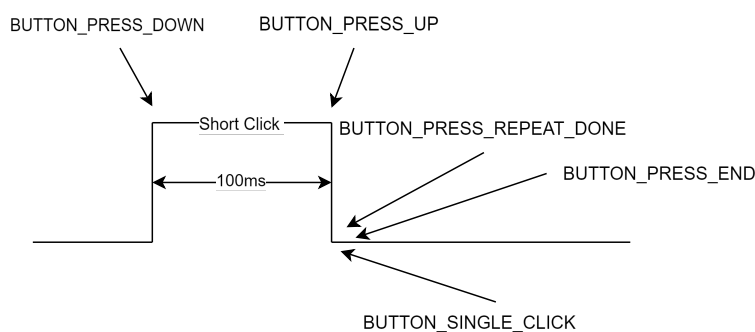
每个按键可以有 **回调**和 **轮询**两种使用方式：

- 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。并不是所有的按键事件都会及时的拿到，存在丢失事件的风险。

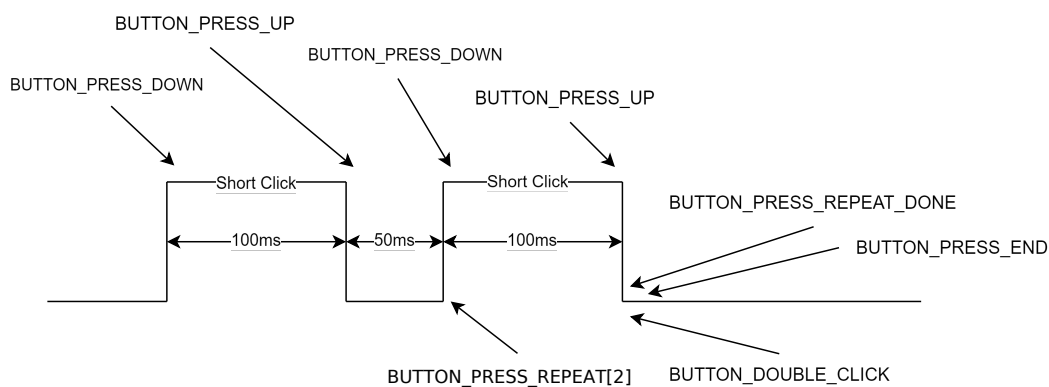
当然你也可以将以上两种方式组合使用。

注意：回调函数中不能有 `TaskDelay` 等阻塞的操作

Single Click

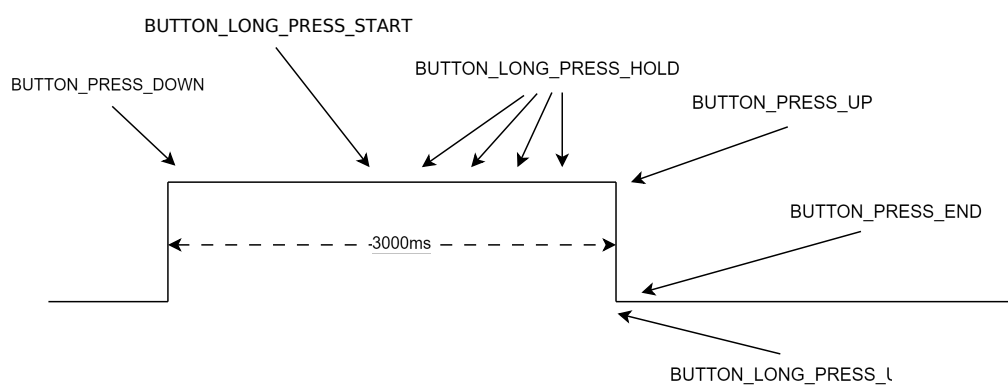


Double Click



LongPress

`BUTTON_LONG_PRESS_TIME_MS`: 1500ms



10.1.2 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 ADC 扫描的样本数
- `BUTTON_LONG_PRESS_HOLD_SERIAL_TIME_MS`: 长按期间触发的 CALLBACK 间隔时间

10.1.3 应用示例

创建按键

```
// create gpio button
const button_config_t btn_cfg = {0};
const button_gpio_config_t btn_gpio_cfg = {
    .gpio_num = 0,
    .active_level = 0,
};
button_handle_t gpio_btn = NULL;
esp_err_t ret = iot_button_new_gpio_device(&btn_cfg, &btn_gpio_cfg, &gpio_btn);
if(NULL == gpio_btn) {
    ESP_LOGE(TAG, "Button create failed");
}

// create adc button
const button_config_t btn_cfg = {0};
button_adc_config_t btn_adc_cfg = {
    .unit_id = ADC_UNIT_1,
    .adc_channel = 0,
    .button_index = 0,
    .min = 100,
    .max = 400,
};

button_handle_t adc_btn = NULL;
esp_err_t ret = iot_button_new_adc_device(&btn_cfg, &btn_adc_cfg, &adc_btn);
if(NULL == adc_btn) {
    ESP_LOGE(TAG, "Button create failed");
}

// create matrix keypad button
const button_config_t btn_cfg = {0};
const button_matrix_config_t matrix_cfg = {
    .row_gpios = (int32_t[]){4, 5, 6, 7},
    .col_gpios = (int32_t[]){3, 8, 16, 15},
    .row_gpio_num = 4,
    .col_gpio_num = 4,
};
button_handle_t matrix_button = NULL;
esp_err_t ret = iot_button_new_matrix_device(&btn_cfg, &matrix_cfg, btns, &matrix_
↪button);
if(NULL == matrix_button) {
    ESP_LOGE(TAG, "Button create failed");
}
```

备注: 当 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, NULL, 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_args_t args = {
    .long_press.press_time = 2000,
};

iot_button_register_cb(gpio_btn, BUTTON_LONG_PRESS_START, &args, ↪
↪button_auto_check_cb_1, NULL);

args.long_press.press_time = 5000;
iot_button_register_cb(gpio_btn, BUTTON_LONG_PRESS_START, &args, ↪
↪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。

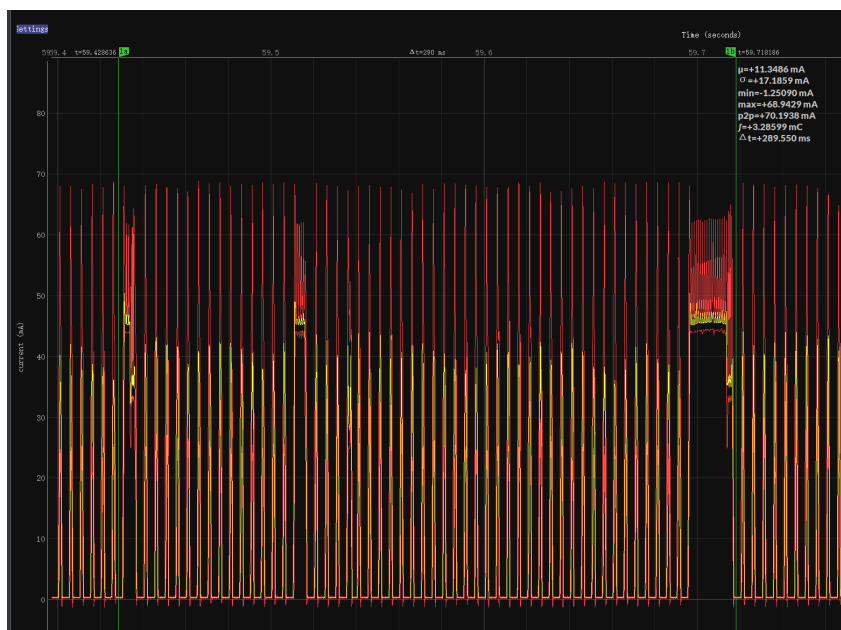
所需配置:

- 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

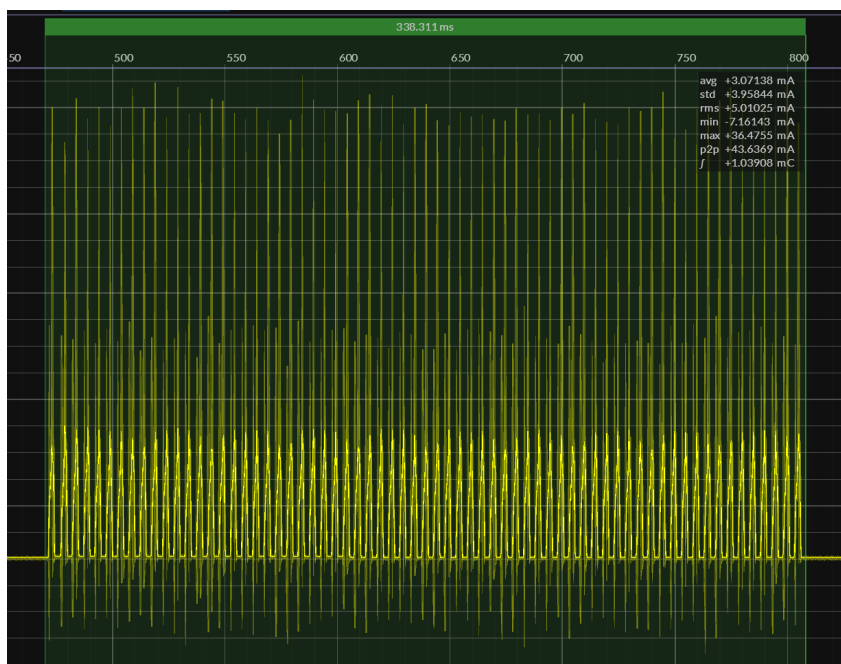
备注: 该功能只保证 `Button` 组件只在使用中才唤醒 `CPU`, 不保证 `CPU` 一定会进入低功耗模式

功耗对比：

- 未开启低功耗模式，按下一次按键

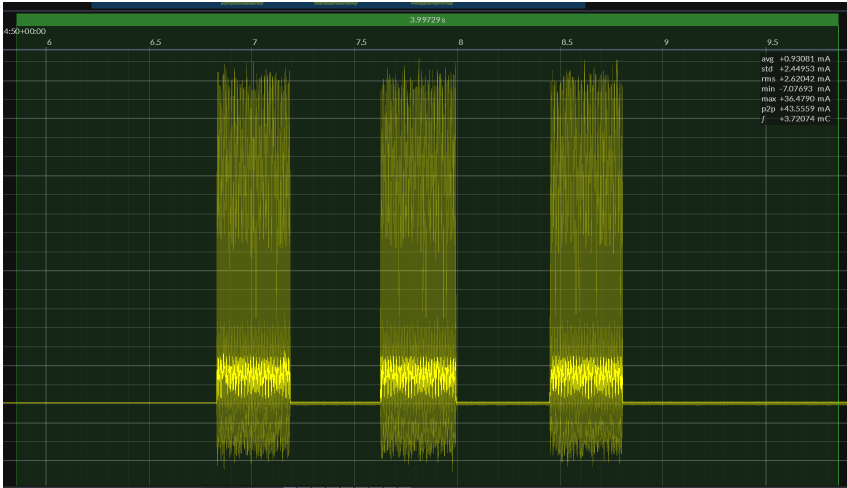
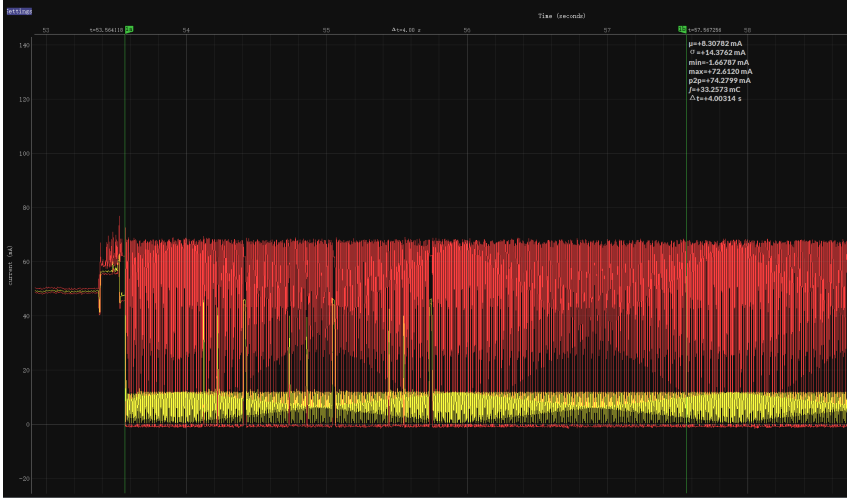


- 开启低功耗模式，按下一次按键



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 = {0};
button_gpio_config_t gpio_cfg = {
    .gpio_num = button_num,
    .active_level = BUTTON_ACTIVE_LEVEL,
    .enable_power_save = true,
};

button_handle_t btn;
iot_button_new_gpio_device(&btn_cfg, &gpio_cfg, &btn);
```

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 CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP 选项后，如何正常使用按键？

- 开启这个宏后，GPIO 模块会下电，如果需要使用按键功能，必须选用 RTC/LP GPIO，并将唤醒源修改为 EXT 1

GPIO 类型	是否开启 CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP	唤醒源
数字管脚	N	GPIO 电平触发
数字管脚	Y	无
RTC/LP 管脚	N	GPIO 电平触发 / EXT 1
RTC/LP 管脚	Y	EXT 1

备注： ESP32-C5, ESP32-C6 的 LP GPIO 可以支持 GPIO 电平唤醒和 EXT 1 唤醒，同时也需要开启 gpio_hold_en

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();
```

10.1.4 API Reference

Header File

- components/button/include/iot_button.h

Functions

`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_event_args_t](#) *event_args, [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
- **`event_args`** –Button event arguments
- **`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_cb` (button_handle_t btn_handle, [button_event_t](#) event, [button_event_args_t](#) *event_args)

Unregister all the callbacks associated with the event.

参数

- **`btn_handle`** –A button handle to unregister
- **`event`** –Button event
- **`event_args`** –Used for unregistering a specific callback.

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_cb` (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`

`const char *iot_button_get_event_str (button_event_t event)`

Get the string representation of a button event.

This function returns the corresponding string for a given button event. If the event value is outside the valid range, the function returns error string “event value is invalid” .

参数 **event** –[in] The button event to be converted to a string.

返回

- Pointer to the event string if the event is valid.
- ” invalid event” if the event value is invalid.

`esp_err_t iot_button_print_event (button_handle_t btn_handle)`

Log the current button event as a string.

This function prints the string representation of the current event associated with the button.

参数 **btn_handle** –[in] Handle to the button object.

返回

- ESP_OK: Successfully logged the event string.
- ESP_FAIL: Invalid button handle.

`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.

`uint32_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

esp_err_t **iot_button_register_power_save_cb** (const *button_power_save_config_t* *config)

Register a callback function for power saving. The config->enter_power_save_cb function will be called when all keys stop working.

参数 config –Button power save config

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE No button registered
- ESP_ERR_INVALID_ARG Arguments is invalid
- ESP_ERR_NO_MEM Not enough memory

Unions

union **button_event_args_t**

#include <iot_button.h> Button events arg.

Public Members

struct *button_event_args_t::long_press_t* **long_press**

long press struct, for event BUTTON_LONG_PRESS_START and BUTTON_LONG_PRESS_UP

struct *button_event_args_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_power_save_config_t**

Structs to store power save callback info.

Public Members

button_power_save_cb_t **enter_power_save_cb**

Callback function when entering power save mode

void ***usr_data**

User data for the callback

Type Definitions

typedef void (***button_cb_t**)(void *button_handle, void *usr_data)

typedef void (***button_power_save_cb_t**)(void *usr_data)

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_param_t**

Button parameter.

Values:

enumerator **BUTTON_LONG_PRESS_TIME_MS**

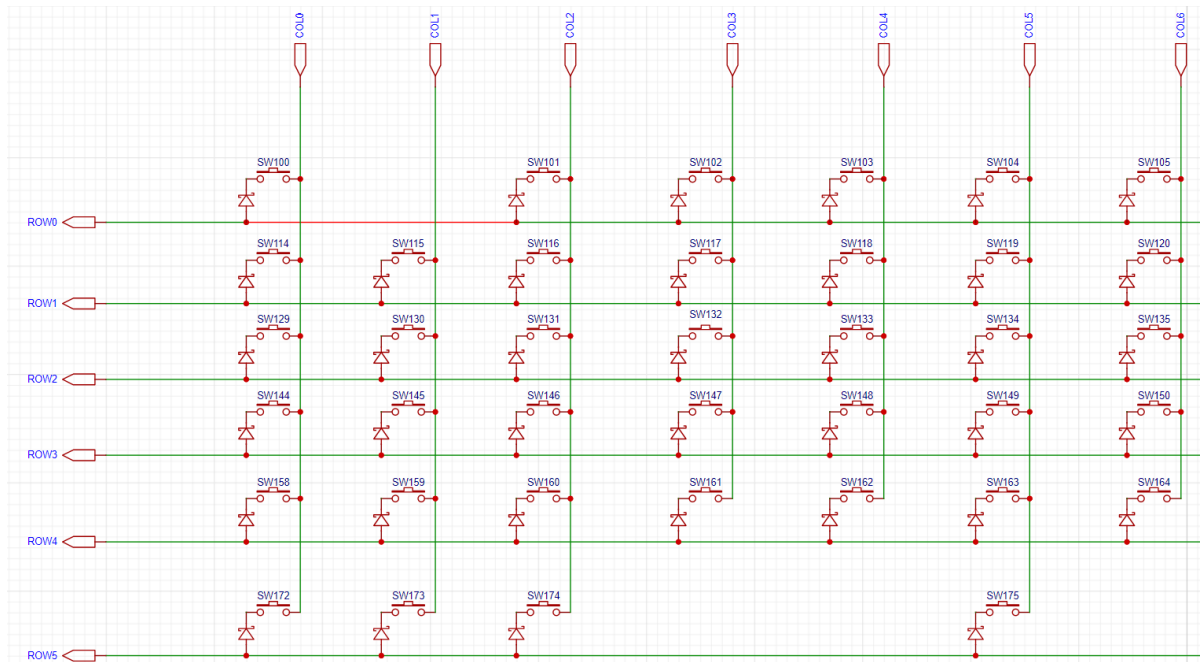
enumerator **BUTTON_SHORT_PRESS_TIME_MS**

enumerator **BUTTON_PARAM_MAX**

10.2 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。

10.2.1 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。

10.2.2 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},
    },
};
```

(下页继续)

(续上页)

```
};

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。

10.2.3 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**

10.3 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。

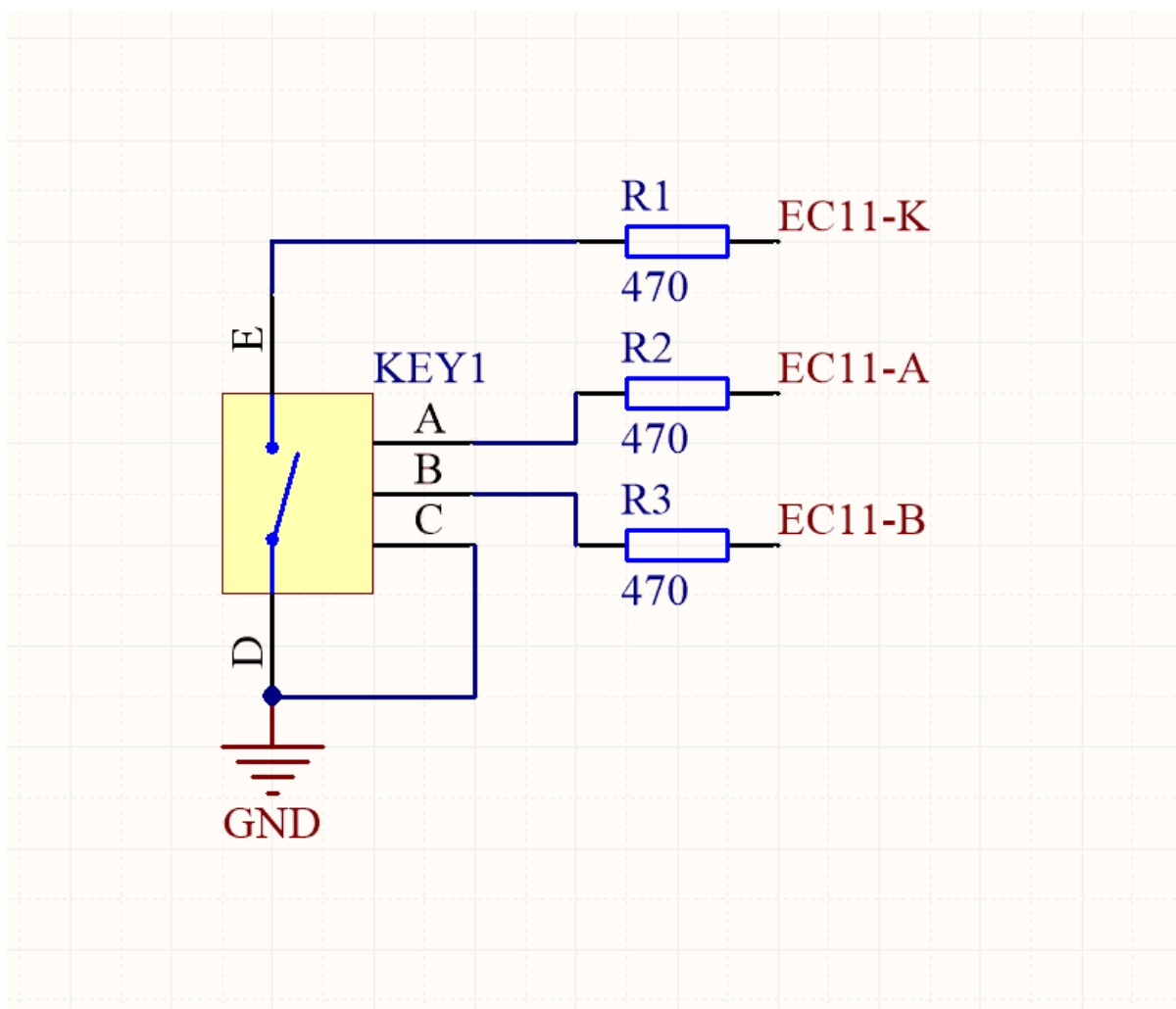
10.3.1 适用的场景

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。

10.3.2 硬件设计

旋转编码器的参考设计如下：



10.3.3 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

10.3.4 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

10.3.5 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

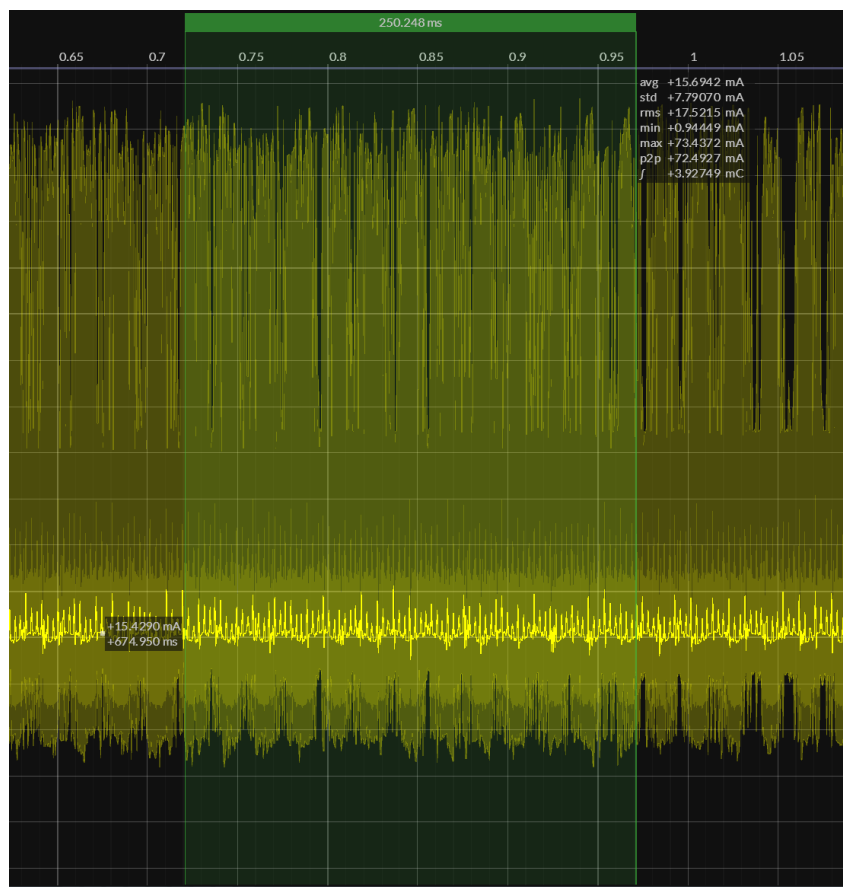
在 light_sleep 模式下，esp_timer 定时器会唤醒 CPU，导致功耗居高不下，Knob 组件提供了通过 GPIO 电平唤醒的低功耗方案。

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

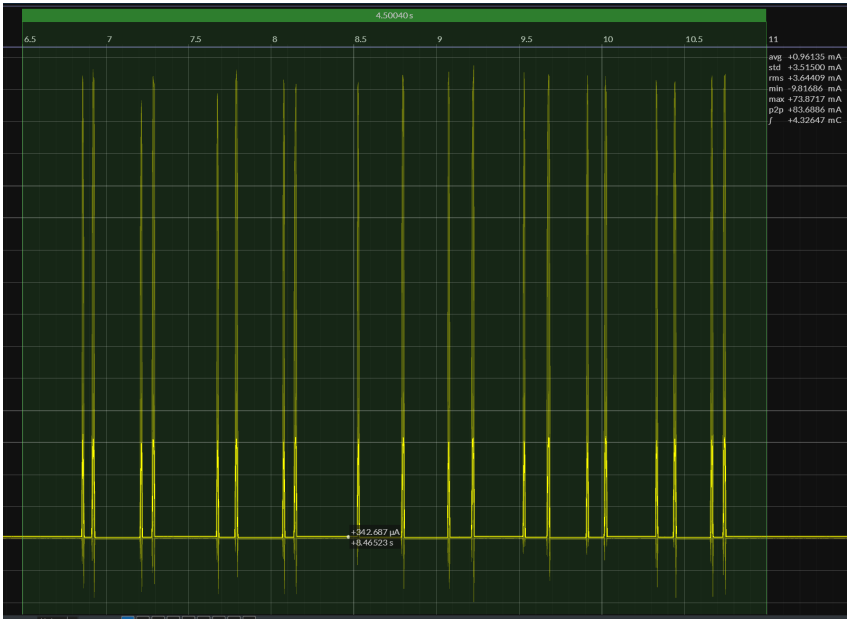
组件支持在任意时刻开启和关闭。

```
// stop knob
iot_knob_stop();
// resume knob
iot_knob_resume();
```

10.3.6 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 cout 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

10.4 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。

10.4.1 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。

10.4.2 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。

10.4.3 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。

10.4.4 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);
}
```

10.4.5 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 11

红外

11.1 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。

11.1.1 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);
}

```

11.1.2 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 12

低功耗方案

12.1 ESP32-P4 备用电池供电方案

ESP32-P4 集成 RTC (Real-Time Clock, 实时时钟) 功能, 包含备用电源输入端 (VBAT), 32.768KHz 外部晶振输入, 电源切换控制等, 可在主电源断开后保持计时准确。

警告: ESP-IDF 尚未完全支持 VBAT 方案, 当前方案仅供测试。请按照如下过程在 master 分支上修改 ESP-IDF 驱动。

支持以下特性:

- 支持主电源掉电保持计时功能, 掉电后由备用电池 (VBAT) 供电
- 支持睡眠/唤醒时 PMU (Power management unit, 电源管理单元) 自动切换电源
- 支持备用电池电压检测, 阈值报警
- 支持备用电池充电, 当 VDDA 存在时, 可通过内部电阻为 VBAT 上的备用电池充电

12.1.1 电池连接与电路保护

ESP32-P4 备用电池的连接非常简单。VBAT 引脚内置充电保护电路, 因此无需外部电阻与二极管, 有效降低了在备用电池通路上增加元件带来的有害风险。

备用电池的选择

- 低自放电率: 电池应具有出色的抗漏电性
- 充放电性能: 较高充放电次数的电池可以提升产品使用寿命
- 放电电流: VBAT 供电功耗较低 (几 μA), 因此大部分电池放电能力均可满足
- 充电电流: 支持 VDDA 给 VBAT 上备用电池充电。软件可配置充电限流电阻 $1\text{K}\Omega \sim 9\text{K}\Omega$

警告: 严禁接入 3.6V 的可充电电池, 由于 3.6V 的可充电电池在满电情况下的电压能够达到 4V 左右, 而 VBAT 引脚的输入电压范围为 2.3V ~ 3.6V, 可能会导致 VBAT 引脚损坏。

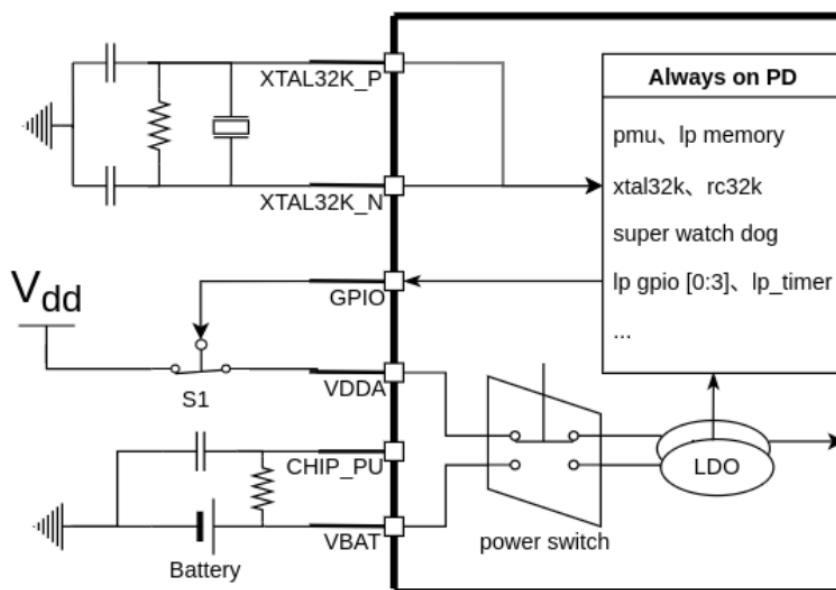


图 1: ESP32-P4 备用电池连接示意图

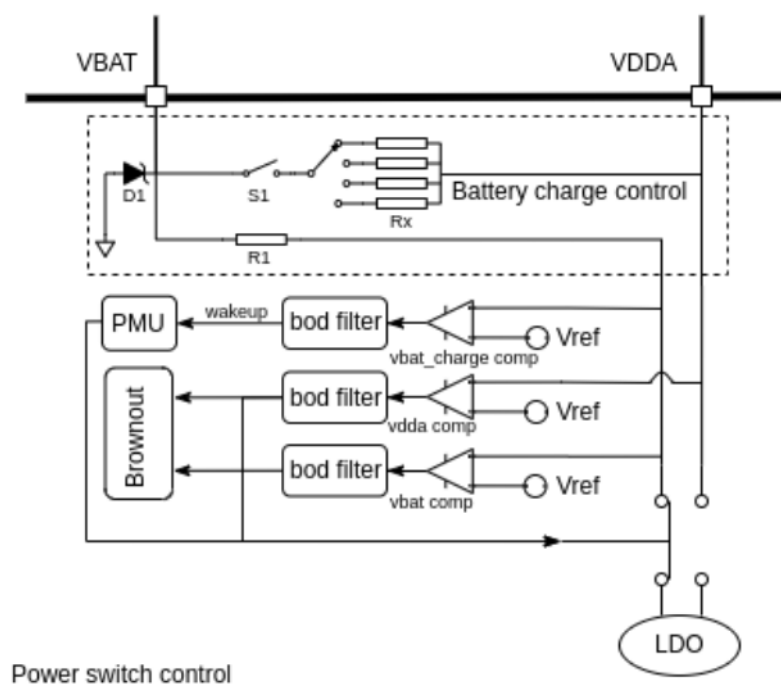


图 2: ESP32-P4 电池切换控制框图

电池上的去耦电容

在电路设计中，用户通常会为每一个供电引脚添加一个去耦电容。但当 VBAT 作为备用电源输入时，电容上的泄露将导致电池寿命明显缩短，此外，电池具有良好的电容特性，因此无需外部去耦电容。

备注： D1 与 R1 为 ESD 保护器件，Rx 为充电保护电阻，使充电电流限制在 1 mA 以下，保护电路满足 UL 认证测试要求。

12.1.2 电源切换

VBAT 与 VDDA 的电源选择由 PMU 在睡眠与唤醒时自动控制或手动选择。用户需要提前配置寄存器。当芯片处于 sleep 状态时可配置 VDDA 或 VBAT 作为电源输入。

- PMU_HP_SLEEP_VDDBAT_MODE: 选择 HP_SLEEP 状态时的供电电源。0: VDDA 供电、1: VBAT 供电、2: 自动选择
- PMU_VDDBAT_SW_UPDATE: 置为 1 使 PMU_VDDBAT_CFG_REG 寄存器配置生效
- PMU_LP_ANA_WAIT_TARGET: 设置唤醒过程中的 delay 时间。时间长度应该大于 VDDA 外部电源的上电稳定时间

备注： 当前 ESP-IDF 尚未完整支持 PMU 驱动。若您需要测试 VBAT 功能，请按照下述测试过程在 ESP-IDF 中添加必要的代码。

测试过程如下：

1. 在 components/esp_hw_support/port/esp32p4/pmu_sleep.c 中添加必要的头文件

```
#include "soc/pmu_reg.h"
#include "soc/lp_analog_peri_reg.h"
#include "soc/lp_gpio_struct.h"
#include "esp_private/regi2c_ctrl.h"
#include "soc/lp_analog_peri_reg.h"
#include "soc/lp_analog_peri_struct.h"
#include "esp_private/regi2c_ctrl.h"
#include "soc/regi2c_bias.h"
```

2. 在 components/esp_hw_support/port/esp32p4/pmu_sleep.c 的 pmu_sleep_start 函数中的 pmu_ll_hp_set_sleep_enable(PMU_instance()->hal->dev); 上方添加如下代码：

方式 1: VDDA 与 VBAT 自动切换方式：

```
#define VBAT_MODE_VDDA 0
#define VBAT_MODE_VBAT 1
#define VBAT_MODE_AUTO 2

REG_SET_FIELD(PMU_HP_SLEEP_LP_DIG_POWER_REG, PMU_HP_SLEEP_VDDBAT_MODE, VBAT_MODE_
↪AUTO);
REG_SET_BIT(PMU_VDDBAT_CFG_REG, PMU_VDDBAT_SW_UPDATE);
while(VBAT_MODE_AUTO != REG_GET_FIELD(PMU_VDDBAT_CFG_REG, PMU_ANA_VDDBAT_MODE));
```

方式 2: VDDA 与 VBAT 手动设置：

```
#define VBAT_MODE_VDDA 0
#define VBAT_MODE_VBAT 1
#define VBAT_MODE_AUTO 2

REG_SET_FIELD(PMU_HP_SLEEP_LP_DIG_POWER_REG, PMU_HP_SLEEP_VDDBAT_MODE, VBAT_MODE_
↪VDDA);
```

(下页继续)

(续上页)

```

REG_SET_BIT(PMU_VDDBAT_CFG_REG, PMU_VDDBAT_SW_UPDATE);
while(VBAT_MODE_VDDA != REG_GET_FIELD(PMU_VDDBAT_CFG_REG, PMU_ANA_VDDBAT_MODE));
REG_SET_FIELD(PMU_LP_SLEEP_LP_DIG_POWER_REG, PMU_LP_SLEEP_VDDBAT_MODE, VBAT_MODE_
↪VBAT);
REG_SET_FIELD(PMU_SLP_WAKEUP_CNTL5_REG, PMU_LP_ANA_WAIT_TARGET, 0xFF);

```

备注：经过测试，deep sleep 下的 VBAT 平均电流为 7 uA，light sleep 下的 VBAT 平均电流为 21 uA

备用电池运行时间

以 CR2032 电池（225 mAh）为例，在 deep sleep 下的理论待机时间为 3.669 年。

12.1.3 欠压检测器 & BOD filter

ESP32-P4 的欠压检测器支持检测 VDDA 与 VBAT 的电压，在电压快速下落至预设阈值以下触发信号，并进行相应的处理。

阈值配置与电压关系：

阈值配置	电压
0	2.52v
1	2.57v
2	2.63v
3	2.68v
4	2.74v
5	2.78v
6	2.83v
7	2.89v

新建测试工程，并添加如下代码检测欠压标志位：

```

REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFL_VDDA, 6);
↪ // Vdda 比较器电压阈值设置 2.83
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFH_VDDA, 7);
↪ // Vdda 比较器电压阈值设置 2.89
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFL_VBAT, 6);
↪ // Vbat 比较器电压阈值设置 2.83
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFH_VBAT, 7);
↪ // Vbat 比较器电压阈值设置 2.89

REG_SET_FIELD(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪UNDERVOLTAGE_TARGET, 20); // 设置bod
↪filter的时间阈值，单位20MHz cycle
REG_SET_FIELD(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪UPVOLTAGE_TARGET, 10); // 设置bod
↪filter的时间阈值，单位20MHz cycle
REG_SET_BIT(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪CNT_CLR);
esp_rom_delay_us(30);
REG_CLR_BIT(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪CNT_CLR);

while (1)
{
    printf("value:%lx\n", REG_READ(LP_ANALOG_PERI_VDD_SOURCE_CNTL_REG));
}

```

(下页继续)

(续上页)

```

REG_WRITE(LP_ANALOG_PERI_VDD_SOURCE_CNTL_REG, 0X30020000);
↪ // 清空VBAT欠压标志位
REG_WRITE(LP_ANALOG_PERI_VDD_SOURCE_CNTL_REG, 0X30040000);
↪ // 清空VDDA欠压标志位
vTaskDelay(500 / portTICK_PERIOD_MS);
}

```

12.1.4 电池充电电路

当选择可充电电池作为 VBAT 电源输入时，芯片支持检测电池电压，并在低于设定阈值通过 VDDA 反向给 VBAT 端电池充电。

充电检测阈值与电压关系：

阈值配置	电压
0	2.52v
1	2.57v
2	2.63v
3	2.68v
4	2.74v
5	2.78v
6	2.83v
7	2.89v

新建测试工程，并添加如下代码检测欠压标志位：

```

REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFL_VDDA, 6);
↪ // Vdda 比较器电压阈值设置 2.83
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFH_VDDA, 7);
↪ // Vdda 比较器电压阈值设置 2.89
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFL_VBAT, 6);
↪ // Vbat 比较器电压阈值设置 2.83
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFH_VBAT, 7);
↪ // Vbat 比较器电压阈值设置 2.89

REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFL_VBAT_CHARGER, 6);
↪ // 充电检测比较器的阈值设置，应高于↪
↪ brownout 报警电压 2.83
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_DREFH_VBAT_CHARGER, 7);
↪ // 充电检测比较器的阈值设置，应高于↪
↪ brownout 报警电压
REGI2C_WRITE_MASK(I2C_BIAS, I2C_BIAS_OR_FORCE_PU_VBAT_CHARGER, 1);
↪ // vbat_charger 比较器强制上电

REG_SET_FIELD(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪ UNDERVOLTAGE_TARGET, 20); // 设置bod↪
↪ filter的时间阈值，单位20MHz cycle
REG_SET_FIELD(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪ UPVOLTAGE_TARGET, 10); // 设置bod↪
↪ filter的时间阈值，单位20MHz cycle
REG_SET_BIT(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪ CNT_CLR);
esp_rom_delay_us(30);
REG_CLR_BIT(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_ANALOG_PERI_VDDBAT_CHARGE_
↪ CNT_CLR);

while(1)
{

```

(下页继续)

(续上页)

```
printf("value:%lx\n", REG_GET_BIT(LP_ANALOG_PERI_VDDBAT_CHARGE_CNTL_REG, LP_
↪ANALOG_PERI_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG)); // 获取 VBAT 管脚充电标志信号
vTaskDelay(500 / portTICK_PERIOD_MS);
}
```

12.1.5 VBAT 供电场景下的外设支持功能

RTC TIMER

在进入休眠状态切换 VBAT 供电后，VDDA 下电后 RTC TIMER 正常计数，再次重启后时间显示正常。测试代码如下：

```
time_t now;
char strftime_buf[64];
struct tm timeinfo;
time(&now);
localtime_r(&now, &timeinfo);
strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
printf("The current date/time is: %s\n", strftime_buf);

const gpio_config_t config = {
    .pin_bit_mask = BIT(GPIO_NUM_0),
    .mode = GPIO_MODE_INPUT,
};

ESP_ERROR_CHECK(gpio_config(&config));
esp_deep_sleep_enable_gpio_wakeup(BIT(GPIO_NUM_0), 0);

esp_deep_sleep_start();
```

备注：您需要按照上述电源切换章节在 ESP-IDF 中添加电源切换代码，同时，测试过程在 VDDA 下电后等待了一段时间后重新上电，并使用 GPIO0 唤醒 ESP32-P4，对比休眠前后时间变化。

LP GPIO/ADC/UART 外设

请参考 ESP-IDF 中 examples/system/ulp/lp_core 的测试驱动进行测试，当前 GPIO/ADC/UART 的测试已支持 ESP32-P4。

Chapter 13

传感器集

13.1 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。

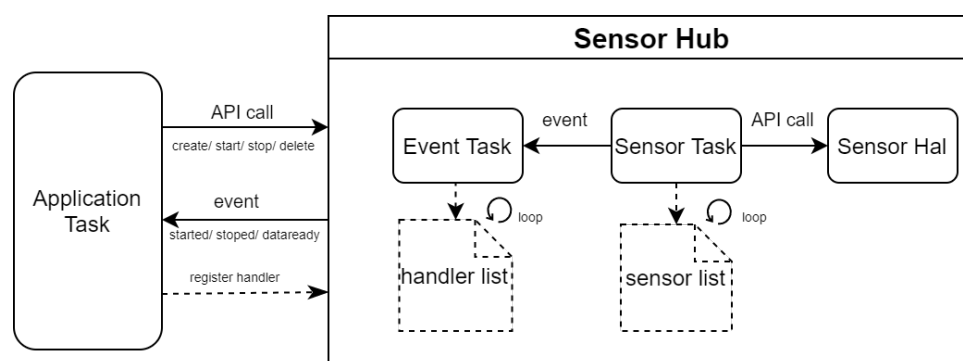


图 1: Sensor Hub 编程模型

Sensor Hub 支持以组件方式加载传感器驱动，用户在添加传感器时只需填入注册到 Sensor Hub 的传感器名称即可直接获取对应的传感器驱动。此外，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。

13.1.1 Sensor Hub 使用方法

sensor_hub 采用 [链接器脚本生成机制](#) 将传感器驱动注册到特定的目标文件段中。对于应用开发者而言，无需关注传感器驱动的具体实现，只需添加对应的传感器组件即可加载对应的驱动。

驱动开发者：

以 ShT3X 温湿度传感器为例，驱动开发者需要将与传感器相关的操作填入到 humiture_impl_t，并创建传感器检测函数。sensor_hub 提供了用于传感器注册接口：SENSOR_HUB_DETECT_FN，驱动开

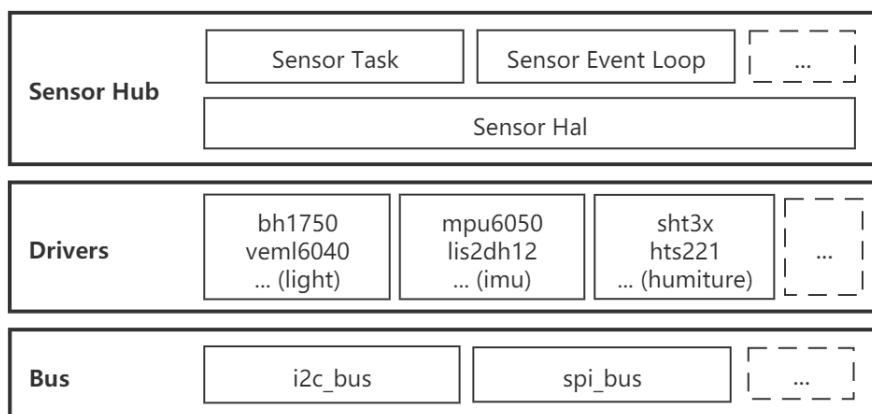


图 2: Sensor Hub 驱动

发者可以直接将对应的函数填入到注册接口中。

```
static humiture_impl_t sht3x_impl = {
    .init = humiture_sht3x_init,
    .deinit = humiture_sht3x_deinit,
    .test = humiture_sht3x_test,
    .acquire_humidity = humiture_sht3x_acquire_humidity,
    .acquire_temperature = humiture_sht3x_acquire_temperature,
};

SENSOR_HUB_DETECT_FN(HUMITURE_ID, sht3x, &sht3x_impl);
```

同时，在组件的 CMakeLists.txt 中添加接口依赖：

```
target_link_libraries(${COMPONENT_LIB} INTERFACE "-u humiture_sht3x_init")
```

应用开发者：

1. 在工程的 idf_component.yml 中添加 sensor_hub 与所需的传感器组件。
2. 创建一个传感器实例：使用 `iot_sensor_create()` 创建一个传感器实例，参数包括传感器名称、传感器配置项和传感器句柄指针。传感器名称用于查找和加载注册到 sensor_hub 中的传感器的驱动，若传感器支持地址可配置，则可以多次创建该名称的传感器。配置项中 bus 用于指定传感器挂载到的总线位置；addr 用于指定传感器对应的地址；type 用于指定传感器对应的类型；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；

```
sensor_config_t sht3x_config = {
    .bus = i2c0_bus_handle,
    .addr = 0x44,
    .mode = MODE_POLLING,
    .min_delay = SENSOR_PERIOD,
};

iot_sensor_create("sht3x", &sht3x_config, &sht3x_handle)
```

3. 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
 - 使用 `iot_sensor_handler_register_with_type()` 通过传感器类型注册回调函数
4. 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；

5. 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；
6. 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
7. 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。

13.1.2 示例程序

1. 温湿度传感器控制 LED 开关示例：[sensors/sensor_control_led](#)。
2. 传感器监测示例：[sensors/sensor_hub_monitor](#)。

13.1.3 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

`const char *sensor_name`
sensor name

`sensor\_type\_t sensor_type`
sensor type

`uint8_t sensor_addr`
sensor addr

`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_device_impl_t**

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_STOPED**

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` (const char *sensor_name, const [sensor_config_t](#) *config, [sensor_handle_t](#) *p_sensor_handle)

Create a sensor instance with specified name and desired configurations.

参数

- **sensor_name** –sensor' s name. Sensor information will be registered by `ESP_SENSOR_DETECT_FN`
- **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_STOPPED` 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** –sensor handle, will set to NULL if delete succeed.

返回

- `ESP_OK` Success
- `ESP_FAIL` Fail

`int iot_sensor_scan` ()

Scan for valid sensors registered in the system.

返回 int number of valid sensors

`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, written by the driver developer.

Public Members

const char ***name**

sensor name

sensor_type_t **sensor_type**

sensor type

struct **sensor_config_t**

sensor initialization parameter

Public Members

bus_handle_t **bus**

i2c/spi bus handle

uint8_t **addr**

set addr of sensor

sensor_type_t **type**

sensor type

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

struct **sensor_hub_detect_fn_t**

Detection function interface for sensors.

Public Members

void *(***fn**)(*sensor_info_t* *sensor_info)

Function pointer for sensor detection.

This function detects a sensor based on the provided sensor information.

Param sensor_info Pointer to the sensor information structure

Return void* Pointer to the detected sensor instance, or NULL if detection failed

Macros

SENSOR_HUB_DETECT_FN (type_id, name_id, impl)

Defines a sensor detection function to be executed automatically in the application layer.

This macro defines a function and its corresponding metadata to facilitate automatic sensor detection and initialization. The function must return the implementation (`impl`) on success; otherwise, an error is logged, and the automatic detection process in the application layer will be aborted.

To prevent the linker from optimizing out the sensor driver, the driver must include at least one undefined symbol that is explicitly referenced during the linking phase. This ensures that the linker retains the driver, even if no other files directly depend on its symbols.

For example, in the sensor driver's `CMakeLists.txt`, include the following to force the linker to keep the required symbol:

```
target_link_libraries(${COMPONENT_LIB} INTERFACE "-u <symbol_name>" )
```

Replace `<symbol_name>` with an appropriate symbol, such as `sht3x_init`, defined in the driver.

参数

- **type_id** –The sensor type identifier.
- **name_id** –The unique name identifier for the sensor.
- **impl** –The implementation returned on successful detection.

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
```

13.2 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。

13.2.1 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	规格书	
HTS221	温度、湿度	I2C	ST	规格书	√
SHT3X	温度、湿度	I2C	Sensirion	规格书	√
MVH3004D	温度、湿度	I2C	—		

13.2.2 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](#)

13.3 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。

13.3.1 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	规格书	√
MPU6050	3-axis acceler + 3-axis gyro	I2C	InvenSense	规格书	√

13.3.2 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](#)

13.4 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。

13.4.1 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	规格书	√
VEML6040	Light RGBW	I2C	Vishay	规格书	√
VEML6075	Light UVA UVB	I2C	Vishay	规格书	√

13.4.2 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](#)

13.5 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。

13.5.1 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	规格书	

13.6 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。

13.6.1 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	规格书	

13.7 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，其主要的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： $R_t = R * \text{EXP}(B * (1/T_1 - 1/T_2))$ -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

13.7.1 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));
```

13.7.2 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 (Ω)

`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
```

13.8 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。

13.8.1 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	规格书	x

13.8.2 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

ina236_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

ina236_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 (***ina236_alert_cb_t**)(void *arg)

ina236 alert callback function

13.9 功率测量

本示例演示了如何使用 BL0937 功率测量芯片来检测电压、电流、有功功率和能耗等电气参数。使用 FreeRTOS 在 ESP32 上实现, 展示了如何配置 BL0937 功率测量芯片并与开发芯片连接。该示例初始化功率测量系统、获取各种参数并定期记录。

13.9.1 适配列表

名称	功能	供应商	规格书	硬件抽象层
BL0937	检测电量参数, 例如电压、电流以及功率和消耗的电量	BELLING	BL0937 Datasheet	x

13.9.2 API 参考

The following API implements hardware abstraction for power measure. Users can directly call this layer of code to write sensor applications.

Header File

- `components/sensors/power_measure/include/power_measure.h`

Functions

`esp_err_t power_measure_init (power_measure_init_config_t *config)`

power measure component init

参数 **config** –configuration for power measure hardware

返回

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_NO_MEM`
- `ESP_ERR_INVALID_ARG`
- `ESP_ERR_INVALID_STATE`

`esp_err_t power_measure_deinit ()`

power measure component deinit

返回

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t power_measure_get_voltage (float *voltage)`

get current voltage (V)

参数 **voltage** –current voltage (V)

返回

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_INVALID_ARG`
- `ESP_ERR_INVALID_STATE`

`esp_err_t power_measure_get_current (float *current)`

get current current (A)

参数 **current** –current current (A)

返回

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_INVALID_ARG`
- `ESP_ERR_INVALID_STATE`

`esp_err_t power_measure_get_active_power (float *active_power)`

get current active power (W)

参数 **active_power** –current active power (W)

返回

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_INVALID_ARG`
- `ESP_ERR_INVALID_STATE`

`esp_err_t power_measure_get_power_factor (float *power_factor)`

get current power factor

参数 **power_factor** –power factor

返回

- `ESP_OK`
- `ESP_FAIL`

- ESP_ERR_INVALID_ARG
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_get_energy** (float *energy)

get current energy (Kw/h)

参数 energy –Power consumed by the load (Kw/h)

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_start_energy_calculation** ()

start the energy calculation

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_stop_energy_calculation** ()

stop the energy calculation

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_reset_energy_calculation** ()

reset the energy accumulated value in flash

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_start_calibration** (*calibration_parameter_t* *para)

start calibration for factory test

参数 para –calibration parameters

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_get_calibration_factor** (*calibration_factor_t* *factor)

get calibration factor from flash

参数 factor –calibration factor

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_STATE

esp_err_t **power_measure_calibration_factor_reset** (void)

reset factor in flash

返回

- ESP_OK
- ESP_FAIL **POWER_MEASURE**

Structures

struct **overcurrent_event_data_t**

Data structure for overcurrent event data.

Public Members

float **current_value**

Current value measured during the overcurrent event

uint16_t **trigger_value**

Trigger threshold for overcurrent detection

struct **overvoltage_event_data_t**

Data structure for overvoltage event data.

Public Members

float **voltage_value**

Voltage value measured during the overvoltage event

uint16_t **overvalue**

Threshold value for overvoltage detection

struct **undervoltage_event_data_t**

Data structure for undervoltage event data.

Public Members

float **voltage_value**

Voltage value measured during the undervoltage event

uint16_t **undervalue**

Threshold value for undervoltage detection

struct **calibration_parameter_t**

Structure for calibration parameters.

Public Members

float **standard_power**

Standard power in watts

float **standard_voltage**

Standard voltage in volts

float **standard_current**

Standard current in amperes

struct **calibration_factor_t**

Structure for calibration factors.

Public Members

float **ki**

Calibration factor for current

float **ku**

Calibration factor for voltage

float **kp**

Calibration factor for power

struct **chip_config_t**

Configuration structure for the chip.

Public Members

power_measure_chip_t **type**

Type of the chip used

calibration_factor_t **factor**

Calibration factors for the chip

gpio_num_t **sel_gpio**

GPIO number for selection

gpio_num_t **cf1_gpio**

GPIO number for CF1

gpio_num_t **cf_gpio**

GPIO number for CF

uint8_t **pin_mode**

Pin mode configuration

float **sampling_resistor**

Value of the sampling resistor

float **divider_resistor**

Value of the divider resistor

struct **power_measure_init_config_t**

Initialization configuration structure for power measurement.

Public Members

chip_config_t **chip_config**

Configuration for the chip

uint16_t **overcurrent**

Overcurrent threshold

uint16_t **overvoltage**

Overvoltage threshold

uint16_t **undervoltage**

Undervoltage threshold

bool **enable_energy_detection**

Flag to enable energy detection

Macros

STORGE_FACTOR_KEY

STORGE_FACTOR_KEY.

Enumerations

enum **power_measure_event_t**

POWER_MEASURE event base.

Enumeration of power measurement events.

Values:

enumerator **POWER_MEASURE_INIT_DONE**

Initialisation Done

enumerator **POWER_MEASURE_OVERCURRENT**

Overcurrent detected

enumerator **POWER_MEASURE_OVERVOLTAGE**

Overvoltage detected

enumerator **POWER_MEASURE_UNDERVOLTAGE**

Undervoltage

enumerator **POWER_MEASURE_ENERGY_REPORT**

report energy

enumerator **POWER_MEASURE_HOME_APPLIANCES_ONLINE**

appliances online

enumerator **POWER_MEASURE_HOME_APPLIANCES_OFFLINE**

appliances offline

enum **power_measure_rated_voltage_t**

Enumeration of rated voltage levels.

Values:

enumerator **RATED_VOLTAGE_110V**

Rated voltage of 110 volts

enumerator **RATED_VOLTAGE_120V**

Rated voltage of 120 volts

enumerator **RATED_VOLTAGE_220V**

Rated voltage of 220 volts

enum **power_measure_chip_t**

Enumeration of supported chip types.

Values:

enumerator **CHIP_BL0937**

BL0937 chip type

enumerator **CHIP_MAX**

Maximum chip type

13.10 电池电量计

电池电量计是一种用于监测电池、计量电量的传感器，其基本功能为监测电压、充电/放电电流与电池温度，并估计电池荷电状态 (SOC) 以及电池的完全充电容量 (FCC)。

13.10.1 已适配列表

名称	功能	总线	厂商	规格书
MAX17048/MAX17049	提供精确的电池电压、充电状态信息、充放电速率以及可配置报警指示	I2C	Analog Devices	规格书
BQ27220	单节电池电量监测计，采用补偿放电结束电压 (CEDV) 算法进行电量检测	I2C	Texas Instruments	规格书

13.11 ADC 传感器

ADC 传感器是一种将模拟电压信号转换为数字信号的传感器，广泛应用于数据采集和信号处理系统中，实现对电压变化的精准监测和分析。

13.11.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
MCP3201	ADC	SPI	Microchip	规格书	

Chapter 14

触摸传感器

14.1 触摸接近感应传感器

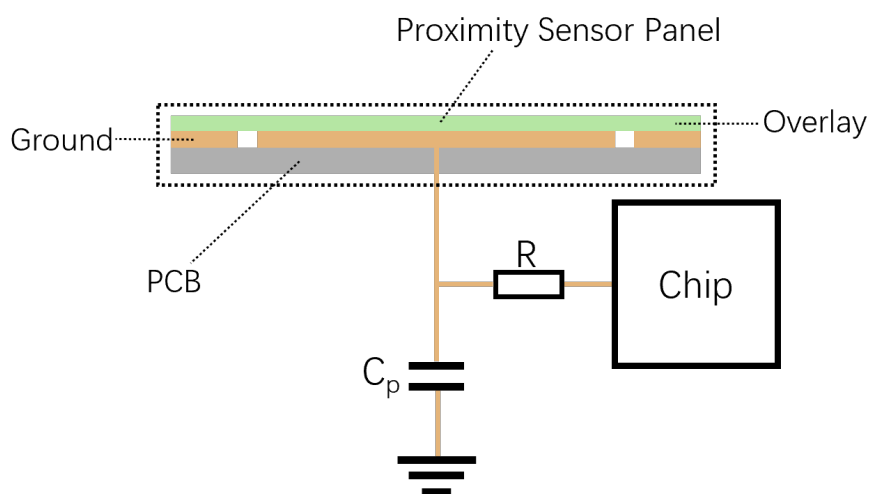
`touch_proximity_sensor` 组件基于 ESP32-S3 内置的触摸传感器进行开发，使用该组件可以轻松实现触摸接近感应功能。

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。
-

14.1.1 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。

14.1.2 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。

14.1.3 配置参考

创建接近感应传感器

使用 `touch_proximity_sensor` 组件，可通过 `touch_proxi_config_t` 结构体来配置接近感应传感器。

```
typedef struct {
    uint32_t channel_num;           /*!< 触摸接近感应通道数量 */
    uint32_t *channel_list;         /*!< 触摸通道列表 */
    float *channel_threshold;       /*!< 每个通道的触发阈值 */
    uint32_t debounce_times;       /*!<
    ↪ 确认状态改变所需的连续读数次数 */
    uint32_t *channel_gold_value;   /*!< 触摸通道参考值 */
    bool skip_lowlevel_init;       /*!<
    ↪ 使用现有触摸驱动时跳过低级初始化 */
} touch_proxi_config_t;
```

主要参数说明：

参数	说明
channel_num	触摸接近感应通道数量，最多支持 3 个
channel_list	触摸通道列表
channel_threshold	每个通道的触发阈值数组
debounce_times	确认状态改变所需的连续读数次数
channel_gold_value	可选的触摸通道参考值
skip_lowlevel_init	使用现有触摸驱动时跳过低级初始化

配置参数后，使用 `touch_proximity_sensor_create()` 创建接近感应传感器：

```
touch_proxi_config_t config = {
    .channel_num = 1,
    .channel_list = channel_list,
    .channel_threshold = channel_threshold,
    .debounce_times = 2,
};

esp_err_t ret = touch_proximity_sensor_create(&config, &sensor_handle, callback_
    ↪func, NULL);
```

事件处理

接近感应传感器需要定期处理事件以更新状态和触发回调。这可以在任务中完成：

```
void proximity_task(void *arg)
{
    while (1) {
        touch_proximity_sensor_handle_events(sensor_handle);
        vTaskDelay(pdMS_TO_TICKS(20));
    }
}

// 创建任务
xTaskCreate(proximity_task, "proximity_task", 2048, NULL, 5, NULL);
```

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(sensor_handle);
```

14.1.4 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` 默认值为 20，数值越大，触摸传感器新数据的更新速率越慢。
- `smooth_coef` 默认值为 0.7，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` 默认值为 0.05，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` 默认值为 0.5，当 `raw - baseline` 的值大于 `baseline * max_p` 时，`raw` 值为异常值，忽略掉。
- `min_n` 默认值为 0.05，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` 默认值为 0.1，和 `noise_n` 默认值为 0.2，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。
- `reset_p` 默认值为 0，用于基线重置正向去抖动，设置为 0 表示禁用。
- `reset_n` 默认值为 50，用于基线重置负向去抖动。

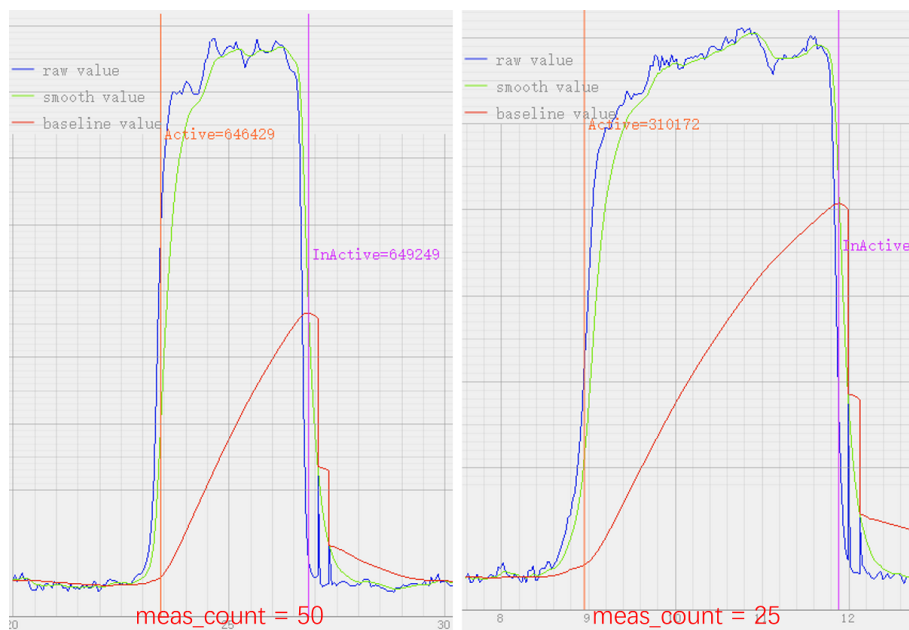
调参波形对比

默认的触摸接近感应传感器配置参数如下：

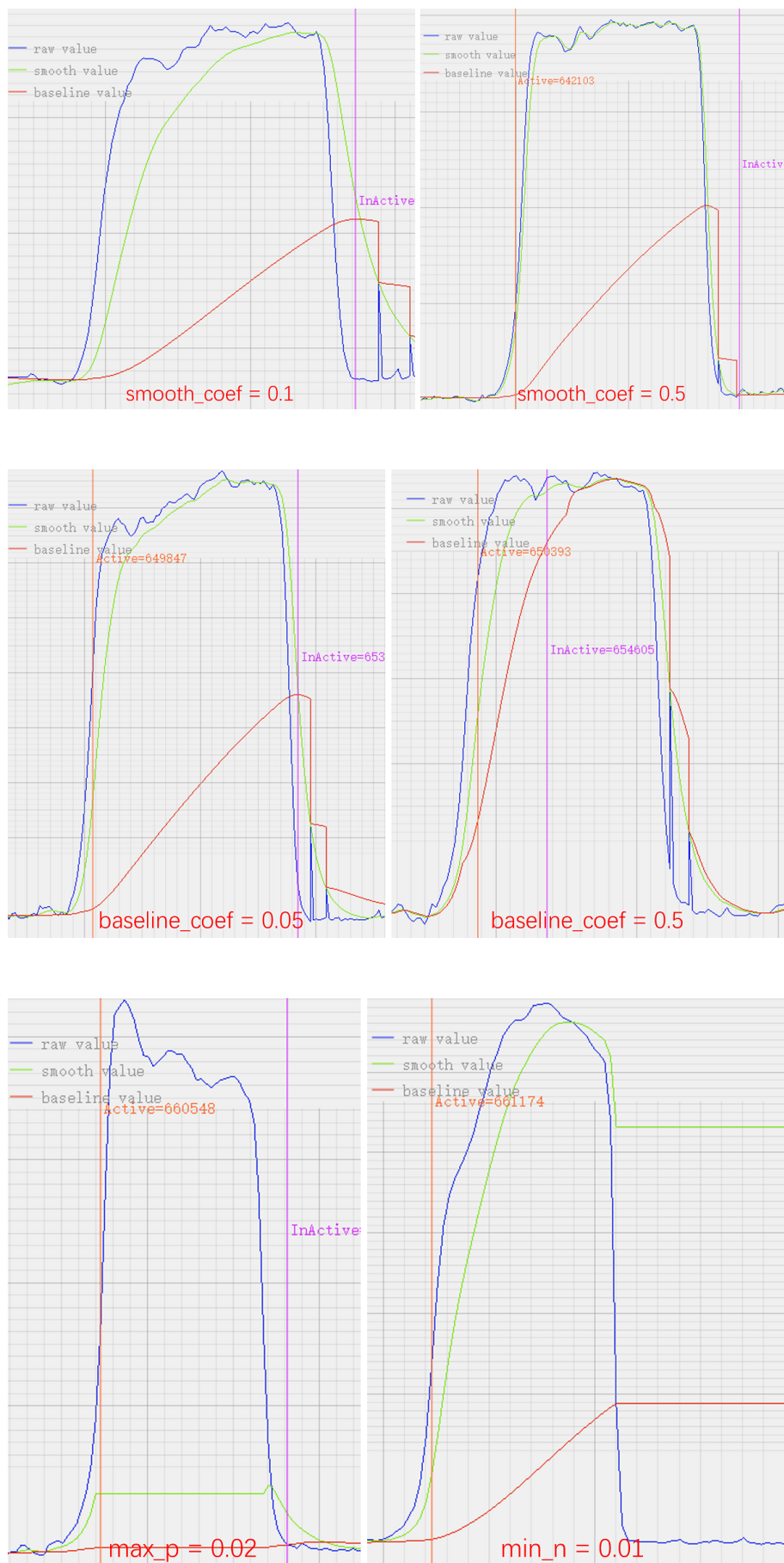
参数	默认值
channel_num	1
channel_list	TOUCH_PAD_NUM8
meas_count	20
smooth_coef	0.7
baseline_coef	0.05
max_p	0.5
min_n	0.05
noise_p	0.1
noise_n	0.2
debounce_p	2
debounce_n	50
reset_p	0
reset_n	50

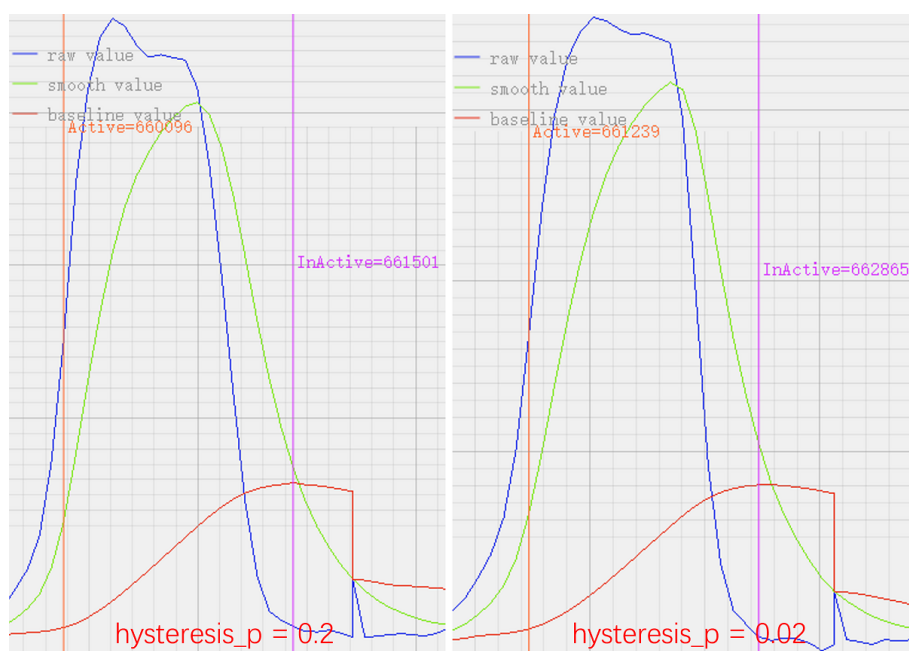
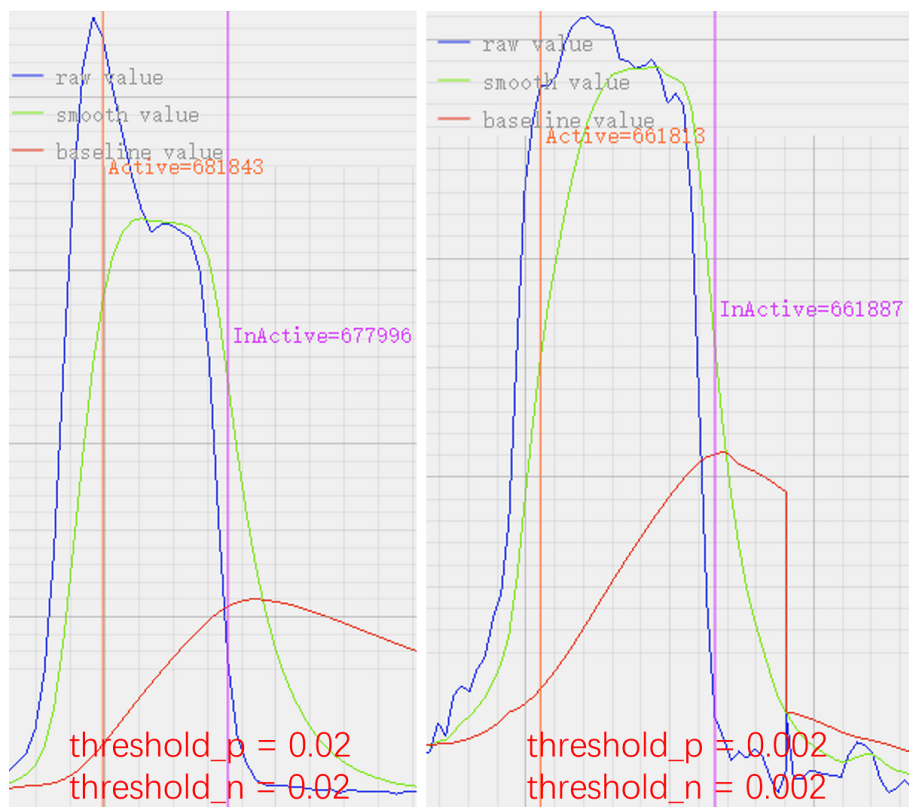
以下调参对比都将在以上参数基础上 **仅修改一个参数** 进行对比。

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：

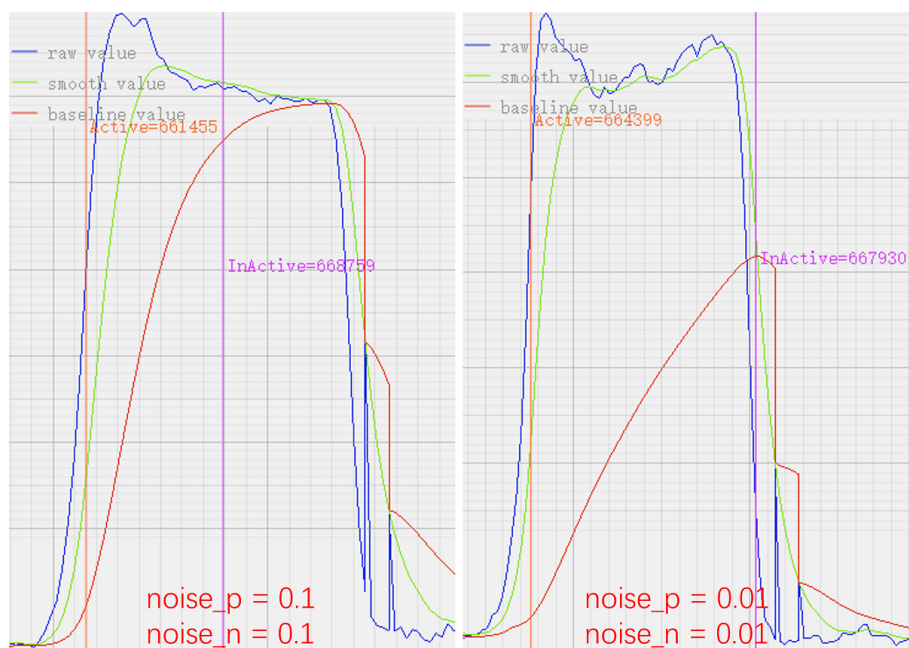


2. 修改 *smooth_coef* 的值，将改变 *smooth* 波形的平滑效果。*smooth_coef* 值越小，平滑效果越好，抗干扰能力越强，*smooth* 跟随 *raw* 越慢，触发响应越慢，反之相反。不同 *smooth_coef* 下的波形对比图如下：
3. 修改 *baseline_coef* 的值，将改变 *baseline* 的更新效果。*baseline_coef* 值越小，*baseline* 跟随 *smooth* 越慢，触发响应越慢，触发的持续时间越长，反之相反。不同 *baseline_coef* 下的波形对比图如下：
4. 修改 *max_p* 和 *min_n* 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：

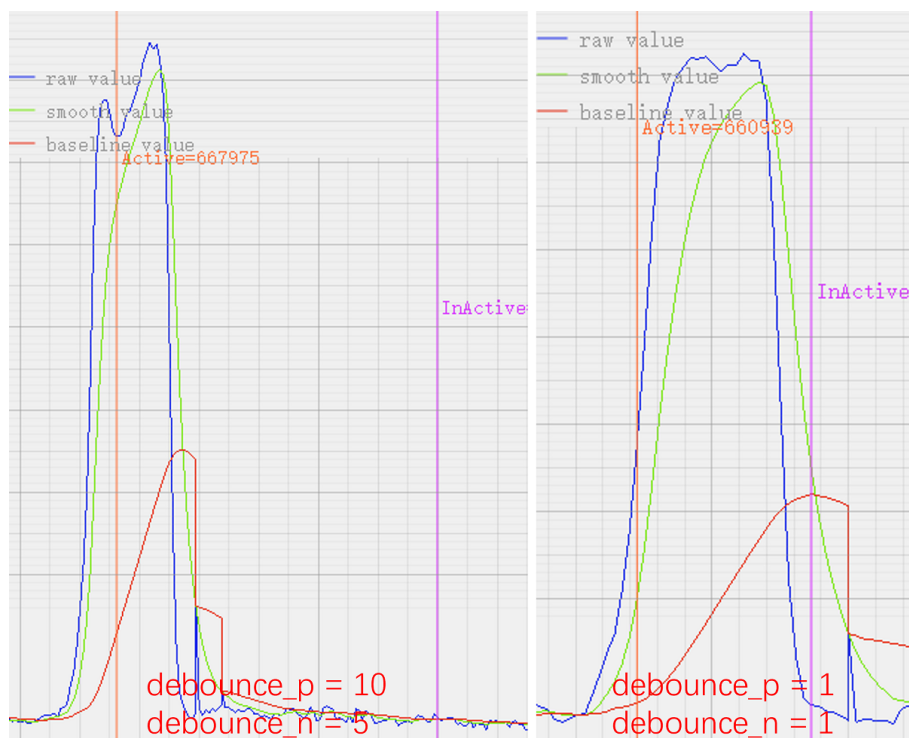




7. 修改 *noise_p* 和 *noise_n* 的值，将改变 *baseline* 的更新效果。*noise_p* 的值越小，*baseline* 跟随 *smooth* 越慢，触发响应越慢，触发的持续时间越长，反之相反。不同 *noise_p* 和 *noise_n* 下的波形对比图如下：



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。

14.1.5 示例程序

- [touch/touch_proximity](#)

14.1.6 API Reference

Header File

- [components/touch/touch_proximity_sensor/include/touch_proximity_sensor.h](#)

Functions

`esp_err_t touch_proximity_sensor_create` (*touch_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_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

`esp_err_t touch_proximity_sensor_get_data` (*touch_proximity_handle_t* handle, uint32_t channel, uint32_t *data)

Get touch proximity sensor data.

Retrieves the smoothed touch sensor reading from the specified channel.

参数

- **handle** –[in] Touch proximity sensor handle
- **channel** –[in] Touch channel number
- **data** –[out] Pointer to store the smoothed touch sensor data

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or data is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not found

`esp_err_t touch_proximity_sensor_get_state` (*touch_proximity_handle_t* handle, uint32_t channel, *proxi_state_t* *state)

Get current state of a proximity channel.

Returns whether the proximity channel is currently considered active (object detected) based on the current sensor readings and configured thresholds.

参数

- **handle** –[in] Touch proximity sensor handle

- **channel** –[in] Touch channel number
- **state** –[out] Pointer to store the channel state

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or state is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not found

esp_err_t **touch_proximity_sensor_handle_events** (*touch_proximity_handle_t* handle)

Handle pending proximity sensor events.

Processes events from the FSM. This function should be called periodically to update proximity states and trigger callbacks.

参数 **handle** –[in] Touch proximity sensor handle

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized

Structures

struct **touch_proxi_config_t**

Configuration structure for touch proximity sensor.

Public Members

uint32_t **channel_num**

Number of touch proximity sensor channels

uint32_t ***channel_list**

Touch channel list

float ***channel_threshold**

Threshold for touch detection for each channel

uint32_t **debounce_times**

Number of consecutive readings needed to confirm state change

uint32_t ***channel_gold_value**

Reference values for touch channels

bool **skip_lowlevel_init**

Skip low level initialization when working with existing touch driver

Type Definitions

typedef struct touch_proximity_sensor_t ***touch_proximity_handle_t**

typedef void (***proxi_cb_t**)(uint32_t channel, *proxi_state_t* event, void *cb_arg)

proximity sensor user callback type

Enumerations

enum **proxi_state_t**

Touch Proximity Sensor State.

Values:

enumerator **PROXI_STATE_INACTIVE**

enumerator **PROXI_STATE_ACTIVE**

14.2 触摸按键传感器

14.2.1 概述

`touch_button_sensor` 组件为 ESP32 系列芯片提供增强型触摸按键检测功能。它使用多频采样（适用于 ESP32-P4）和基于软件 FSM 的处理来提供可靠的触摸检测，即使在嘈杂的环境中也能正常工作。

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。
- 该组件目前适用于 ESP32、ESP32-S2 和 ESP32-S3，并且需要 IDF 版本大于等于 v5.3。

主要特性

- 多频触摸采样，提高抗噪声性能
- 基于 FSM 的触摸检测，具有可配置的阈值
- 支持去抖动以确保状态变化的可靠性
- 支持多个触摸通道
- 基于回调的事件通知

依赖项

- `touch_sensor_fsm`
- `touch_sensor_lowlevel`

14.2.2 配置参数

触摸按键配置结构体

触摸按键可以通过 `touch_button_config_t` 结构体进行配置：

```
typedef struct {
    uint32_t channel_num;           /*!< 触摸按键传感器通道数量 */
    uint32_t *channel_list;         /*!< 触摸通道列表 */
    float *channel_threshold;       /*!< 每个通道的触摸检测阈值 */
    uint32_t *channel_gold_value;   /*!< （可选）触摸通道的参考值 */
}
```

(下页继续)

(续上页)

```

uint32_t debounce_times;          /*!< 确认状态改变所需的连续读数次数 */
bool skip_lowlevel_init;          /*!<
↪ 当与现有触摸驱动程序一起工作时跳过底层初始化 */
} touch_button_config_t;

```

参数说明

参数	描述	默认值
channel_num	触摸按键传感器通道数量	•
channel_list	要使用的触摸通道号数组	•
channel_threshold	每个通道的阈值数组	•
channel_gold_value	触摸通道的参考值（可选）	NULL
debounce_times	确认状态改变所需的连续读数次数	3
skip_lowlevel_init	如果存在触摸驱动则跳过初始化	false

14.2.3 API 使用示例

创建和初始化

```

touch_button_config_t config = {
    .channel_num = 1,
    .channel_list = (uint32_t[]){8},          // 使用触摸通道 8
    .channel_threshold = (float[]){0.02},     // 2% 变化阈值
    .channel_gold_value = NULL,               // 无参考值
    .debounce_times = 3,                     // 3 次读数确认
    .skip_lowlevel_init = false               // 初始化触摸硬件
};
touch_button_handle_t btn_handle = NULL;
esp_err_t ret = touch_button_sensor_create(&config, &btn_handle, button_state_
↪callback, NULL);

```

状态改变回调函数

当触摸按键状态改变时会调用回调函数：

```

void button_state_callback(touch_button_handle_t handle, uint32_t channel, touch_
↪state_t state, void *arg)
{
    switch (state) {
        case TOUCH_STATE_ACTIVE:
            printf("按键通道 %d 被按下\n", channel);
            break;
        case TOUCH_STATE_INACTIVE:
            printf("按键通道 %d 被释放\n", channel);
            break;
    }
}

```

事件处理

触摸按键传感器组件提供了一个事件处理机制，以非阻塞方式处理触摸事件。事件应该在应用程序的主循环或专用任务中定期处理。

```
// 在主循环或任务中
while (1) {
    // 处理所有待处理的触摸事件
    touch_button_sensor_handle_events(btn_handle);

    // 添加延时以防止紧密循环
    vTaskDelay(pdMS_TO_TICKS(20)); // 20ms 间隔通常足够
}
```

14.2.4 示例

- [touch/touch_button_sensor](#)

14.2.5 API 参考

Header File

- [components/touch/touch_button_sensor/include/touch_button_sensor.h](#)

Functions

`esp_err_t touch_button_sensor_create` (*touch_button_config_t* *config, *touch_button_handle_t* *handle, *touch_cb_t* cb, void *cb_arg)

Create a touch button sensor instance.

This function initializes the touch sensor hardware (unless `skip_lowlevel_init` is true), sets up FSM instances for touch detection, and registers callbacks for touch events.

参数

- **config** –Touch button sensor configuration
- **handle** –Pointer to receive the created touch button sensor handle
- **cb** –Callback function for touch state changes
- **cb_arg** –User data to pass to callback function

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if config, handle, or required config fields are NULL
- ESP_ERR_NO_MEM if memory allocation fails
- ESP_FAIL if touch sensor or FSM initialization fails

`esp_err_t touch_button_sensor_delete` (*touch_button_handle_t* handle)

Delete a touch button sensor instance.

Stops all FSMs, unregisters callbacks, frees resources, and optionally deinitializes the touch sensor hardware.

参数 **handle** –Touch button sensor handle to delete

返回

- ESP_OK on success (or if handle is NULL)
- ESP_ERR_INVALID_STATE if sensor state is invalid

`esp_err_t touch_button_sensor_get_data` (*touch_button_handle_t* handle, uint32_t channel, uint32_t channel_alt, uint32_t *data)

Get touch sensor data for a specific channel and frequency.

Retrieves the smoothed touch sensor reading from the specified channel and frequency instance.

参数

- **handle** –Touch button sensor handle
- **channel** –Touch channel number
- **channel_alt** –Frequency instance index (0 to SOC_TOUCH_SAMPLE_CFG_NUM-1)
- **data** –Pointer to store the smoothed touch sensor data

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or data is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not configured

esp_err_t **touch_button_sensor_get_state** (*touch_button_handle_t* handle, uint32_t channel, *touch_state_t* *state)

Get current state of a touch channel.

Returns whether the touch channel is currently considered active (touched) based on the combined state of all frequency instances.

参数

- **handle** –Touch button sensor handle
- **channel** –Touch channel number
- **state** –Pointer to store the channel state

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or state is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not configured

esp_err_t **touch_button_sensor_get_state_bitmap** (*touch_button_handle_t* handle, uint32_t channel, uint32_t *bitmap)

Get active frequency bitmap for a touch channel.

Returns a bitmap indicating which frequency instances have detected a touch on the specified channel. Each bit corresponds to one frequency.

参数

- **handle** –Touch button sensor handle
- **channel** –Touch channel number
- **bitmap** –Pointer to store the frequency activation bitmap

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or bitmap is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not configured

esp_err_t **touch_button_sensor_handle_events** (*touch_button_handle_t* handle)

Handle pending touch sensor events.

Processes events from all FSM instances. This function should be called periodically to update touch states and trigger callbacks.

参数 **handle** –Touch button sensor handle

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized

Structures

struct **touch_button_config_t**

Configuration structure for touch button sensor.

Public Members

uint32_t **channel_num**

Number of touch button sensor channels

uint32_t ***channel_list**

Touch channel list

float ***channel_threshold**

Threshold for touch detection for each channel

uint32_t ***channel_gold_value**

(Optional) Reference values for touch channels

uint32_t **debounce_times**

Number of consecutive readings needed to confirm state change

bool **skip_lowlevel_init**

Skip low level initialization when working with existing touch driver

Type Definitions

typedef struct touch_button_sensor_t ***touch_button_handle_t**

typedef void (***touch_cb_t**)([touch_button_handle_t](#) handle, uint32_t channel, [touch_state_t](#) state, void *cb_arg)

Touch sensor callback function type.

Param handle Touch button sensor handle

Param channel Channel number that triggered the event

Param state Current state of the touch sensor

Param cb_arg User data passed to the callback

Enumerations

enum **touch_state_t**

Touch sensor state.

Values:

enumerator **TOUCH_STATE_INACTIVE**

Touch sensor is inactive

enumerator **TOUCH_STATE_ACTIVE**

Touch sensor is active

14.3 触摸按键传感器

14.3.1 概述

touch_button_sensor 组件为 ESP32 系列芯片提供增强型触摸按键检测功能。

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。
- 该组件目前适用于 ESP32、ESP32-S2 和 ESP32-S3，并且需要 IDF 版本大于等于 v5.3。

主要特性

- 多频触摸采样，提高抗噪声性能
- 支持滑动手势识别
- 基于回调的事件通知

依赖项

- touch_sensor_fsm
- touch_sensor_lowlevel

14.3.2 配置参数

触摸滑动配置结构体

触摸滑动可以通过 `touch_slider_config_t` 结构体进行配置：

```
typedef struct {
    uint32_t channel_num;           /*!< 触摸按键传感器通道数量 */
    uint32_t *channel_list;         /*!< 触摸通道列表 */
    float *channel_threshold;       /*!< 每个通道的触摸检测阈值 */
    uint32_t *channel_gold_value;   /*!< (可选) 触摸通道的参考值 */
    uint32_t debounce_times;       /*!< 确认状态改变所需的连续读数次数 */
    uint32_t filter_reset_times;    /*!< 重置位置的读数次数 */
    uint32_t position_range;        /*!< 滑块位置的最大值，位置范围为 [0, position_
    ↪ range]。数值越大位置精度越高 */
    uint8_t calculate_window;       /*!< 位置计算的窗口大小 (应该 <= channel_
    ↪ num)。设置为 0 则自动默认：2通道用2，3+通道用3 */
    float swipe_threshold;          /*!< 滑动检测的速度阈值 */
    float swipe_hysteresis;         /*!< 滑动检测的速度迟滞 */
    float swipe_alpha;              /*!< 速度滤波参数 */
    bool skip_lowlevel_init;        /*!<
    ↪ 当与现有触摸驱动程序一起工作时跳过底层初始化 */
} touch_slider_config_t;
```


参数说明

参数	描述	默认值
channel_num	触摸按键传感器通道数量	•
channel_list	要使用的触摸通道号数组	•
channel_threshold	每个通道的阈值数组	•
channel_gold_value	触摸通道的参考值（可选）	NULL
debounce_times	确认状态改变所需的连续读数次数	3
filter_reset_times	重置位置的读数次数	•
position_range	滑块位置最大值（数值越大精度越高）	•
calculate_window	位置计算的窗口大小	自动 (0)
swipe_threshold	滑动检测的速度阈值	•
swipe_hysteresis	滑动检测的速度迟滞	•
swipe_alpha	速度滤波参数	•
skip_lowlevel_init	如果存在触摸驱动则跳过初始化	false

计算窗口配置

calculate_window 参数决定了用于位置计算的相邻触摸通道数量。这会影响滑块的精度和抗噪性能。

默认值（当设置为 0 时）：

- **2 通道:** calculate_window = 2（使用所有可用通道）
- **3+ 通道:** calculate_window = 3（精度和抗噪性能之间的最佳平衡）

手动配置：

- 您可以显式设置从 2 到 channel_num 的任何值
- 较小值 (2)：更高的灵敏度但更容易受噪声影响
- 较大值 (3+)：更好的抗噪性能但可能降低分辨率
- 对于高精度应用：推荐值为 $\min(3, \text{channel_num})$
- **注意：**最小值为 2，因为滑动条位置计算至少需要 2 个相邻通道

向后兼容性：

- 设置 calculate_window = 0 启用自动默认选择
- 未初始化此字段的现有代码将使用最优默认值
- 显式设置的值继续按原来的方式工作

14.3.3 API 使用示例

创建和初始化

```
uint32_t channel_list[] = {2, 4, 6, 12, 10, 8};
float threshold[] = {0.005f, 0.005f, 0.005f, 0.005f, 0.005f, 0.005f};
touch_slider_config_t config = {
    .channel_num = 6,
    .channel_list = channel_list,
    .channel_threshold = threshold,
    .filter_reset_times = 5,
    .position_range = 10000,
    .swipe_alpha = 0.9,
    .swipe_threshold = 50,
    .swipe_hysteresis = 40,
    .channel_gold_value = NULL,
    .debounce_times = 0,
    .calculate_window = 0, // 使用默认值（6通道时自动选择为3）
    .skip_lowlevel_init = false
};

// Test successful creation
TEST_ASSERT_EQUAL(ESP_OK, touch_slider_sensor_create(&config, &s_touch_slider,
    ↳touch_slider_event_callback, NULL));
TEST_ASSERT_NOT_NULL(s_touch_slider);
```

事件回调函数

当位置改变或者状态改变时会调用回调函数。结合滑动速度或者松开的位移可以判断手势。

```
static void touch_slider_event_callback(touch_slider_handle_t handle, touch_slider_
    ↳event_t event, int32_t data, void *cb_arg)
{
    if (event == TOUCH_SLIDER_EVENT_RIGHT_SWIPE) {
        printf("右滑 (速度) \n");
    } else if (event == TOUCH_SLIDER_EVENT_LEFT_SWIPE) {
        printf("左滑 (速度) \n");
    } else if (event == TOUCH_SLIDER_EVENT_RELEASE) {
        printf("滑动 %ld\n", data);
        if (data > 1000)
        {
            printf("右滑 (位移) \n");
        }
        else if (data < -1000)
        {
            printf("左滑 (位移) \n");
        }
    } else if (event == TOUCH_SLIDER_EVENT_POSITION)
    {
        printf("位置,%s PRId64 ",%lu\n", get_time_in_ms(), data);
    }
}
```

事件处理

触摸滑动传感器组件提供了一个事件处理机制，以非阻塞方式处理触摸事件。事件应该在应用程序的主循环或专用任务中定期处理。

```
// 在主循环或任务中
while (1) {
    // 处理所有待处理的触摸事件
```

(下页继续)

(续上页)

```
touch_slider_sensor_handle_events(s_touch_slider);

// 添加延时以防止紧密循环
vTaskDelay(pdMS_TO_TICKS(20)); // 20ms 间隔通常足够
}
```

14.3.4 示例

- [touch/touch_slider_sensor](#)

14.3.5 API 参考

Header File

- [components/touch/touch_slider_sensor/include/touch_slider_sensor.h](#)

Functions

`esp_err_t touch_slider_sensor_create` ([touch_slider_config_t](#) *config, [touch_slider_handle_t](#) *handle, [touch_slider_event_cb_t](#) cb, void *cb_arg)

Create a touch slider sensor instance.

This function initializes the touch sensor hardware (unless `skip_lowlevel_init` is true), sets up FSM instances for touch detection, and registers callbacks for touch events.

备注: The `calculate_window` parameter determines how many adjacent channels are used for position calculation. For backward compatibility, if set to 0, default values will be used:

- 2 channels: `window_size` = 2 (use all channels)
- 3+ channels: `window_size` = 3 (optimal balance between precision and noise immunity)
- For high-precision applications: `window_size` = `min(3, channel_num)`

参数

- **config** –Touch slider sensor configuration
- **handle** –Pointer to receive the created touch slider sensor handle
- **cb** –Callback function for touch slider events
- **cb_arg** –User data to pass to callback function

返回

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` if config, handle, or required config fields are NULL, or if `calculate_window` is less than 2 or greater than `channel_num` (0 is acceptable for auto-default)
- `ESP_ERR_NO_MEM` if memory allocation fails
- `ESP_FAIL` if touch sensor or FSM initialization fails

`esp_err_t touch_slider_sensor_delete` ([touch_slider_handle_t](#) handle)

Delete a touch slider sensor instance.

Stops all FSMs, unregisters callbacks, frees resources, and optionally deinitializes the touch sensor hardware.

参数 **handle** –Touch slider sensor handle to delete

返回

- `ESP_OK` on success (or if handle is NULL)
- `ESP_ERR_INVALID_STATE` if sensor state is invalid

esp_err_t **touch_slider_sensor_get_data** (*touch_slider_handle_t* handle, uint32_t channel, uint32_t channel_alt, uint32_t *data)

Get touch sensor data for a specific channel and frequency.

Retrieves the smoothed touch sensor reading from the specified channel and frequency instance.

参数

- **handle** –Touch slider sensor handle
- **channel** –Touch channel number
- **channel_alt** –Frequency instance index (0 to SOC_TOUCH_SAMPLE_CFG_NUM-1)
- **data** –Pointer to store the smoothed touch sensor data

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle or data is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized
- ESP_ERR_NOT_FOUND if channel not configured

void **touch_slider_sensor_get_state** (*touch_slider_handle_t* handle, bool *pressed, uint32_t *pos, float *speed)

Get touch sensor data for a specific channel and frequency.

Retrieves the smoothed touch sensor reading from the specified channel and frequency instance.

参数

- **handle** –Touch slider sensor handle
- **pressed** –Touch slider is pressed or not
- **pos** –Touch slider speed
- **speed** –Touch slider speed

esp_err_t **touch_slider_sensor_handle_events** (*touch_slider_handle_t* handle)

Handle pending touch sensor events.

Processes events from all FSM instances. This function should be called periodically to update touch states and trigger callbacks.

参数 **handle** –Touch slider sensor handle

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if handle is NULL
- ESP_ERR_INVALID_STATE if sensor not initialized

Structures

struct **touch_slider_config_t**

Configuration structure for touch slider sensor.

Public Members

uint32_t **channel_num**

Number of touch slider sensor channels

uint32_t ***channel_list**

Touch channel list

float ***channel_threshold**

Threshold for touch detection for each channel

uint32_t ***channel_gold_value**

(Optional) Reference values for touch channels

uint32_t **debounce_times**

Number of consecutive readings needed to confirm state change

uint32_t **filter_reset_times**

Number of consecutive readings to reset position filter

uint32_t **position_range**

Maximum position value of touch slider, range [0, position_range]. Higher values provide better position resolution

uint8_t **calculate_window**

Window size for position calculation (should be <= channel_num). Set to 0 for auto-default: 2 for 2-channel, 3 for 3+ channels

float **swipe_threshold**

The speed threshold for identifying swiping

float **swipe_hysteresis**

The speed hysteresis for identifying swiping

float **swipe_alpha**

Filter parameter for estimating speed

bool **skip_lowlevel_init**

Skip low level initialization when working with existing touch driver

Type Definitions

```
typedef struct touch_slider_sensor_t *touch_slider_handle_t
```

```
typedef void (*touch_slider_event_cb_t)(touch_slider_handle_t handle, touch_slider_event_t event,  
int32_t data, void *cb_arg)
```

Touch slider sensor event callback function type.

Param handle Touch slider sensor handle

Param event Touch slider event

Param data Touch slider data. Position if event == TOUCH_SLIDER_EVENT_POSITION, displacement if event == TOUCH_SLIDER_EVENT_RELEASE

Param cb_arg User data passed to the callback

Enumerations

```
enum touch_slider_event_t
```

Touch slider event.

Values:

enumerator **TOUCH_SLIDER_EVENT_NONE**

Touch slider is stationary

enumerator **TOUCH_SLIDER_EVENT_RIGHT_SWIPE**

Touch slider is swiped right (from 0 to position_range)

enumerator **TOUCH_SLIDER_EVENT_LEFT_SWIPE**

Touch slider is swiped left (from position_range to 0)

enumerator **TOUCH_SLIDER_EVENT_RELEASE**

Touch slider is released

enumerator **TOUCH_SLIDER_EVENT_POSITION**

Touch slider position is calculated

Chapter 15

存储方案

15.1 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 卡	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 for SD	*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 for eMMC	*2
USB 闪存盘	大容量、可插拔	声音、视频文件存储	GB	USB	12 Mbps/480 Mbps	USB Host Driver	*3
EEPROM	可按字节寻址，低成本	参数存储	MB	I2C	100 ~ 400 kHz	eeprom driver	

备注：

- *1. SD 卡的 SDIO 模式，仅在支持 SDIO 主机外设的 ESP 芯片（如 ESP32、ESP32-S3）上可用；
- *2. 仅在支持 SDIO 主机外设的 ESP 芯片（如 ESP32、ESP32-S3）上可用；
- *3. 仅在支持 USB-OTG 外设的 ESP 芯片（如 ESP32-S2、ESP32-S3）上可用。

15.1.1 SPI Flash

ESP 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。

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，主 Flash 指定空间可以作为非易失性存储（NVS）空间存放应用程序参数，或挂载到文件系统（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）等操作模式，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。

备注：对于 ESP-IDF v4.0 以上版本，SPI Flash 组件不仅支持使用主 Flash 进行用户数据存储，同时可支持外接第二块 Flash 芯片进行数据的存储。

参考文档：

- [SPI Flash API](#)
- [NVS Flash API](#)
- [FatFS API](#)

示例程序：

- [storage/nvs](#)：使用 NVS 进行参数存储；
- [storage/fatfs](#)：使用 FatFS 进行文件存储。

15.1.2 SD 卡

ESP 系列芯片支持使用 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 或高速 40 MHz 等速率模式。需要注意的是 SDIO 接口一般只能使用 [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](#)：提供 SD 卡的初始化接口和协议层 API；
- [SDMMC 主机驱动](#)：提供 SDIO 模式的接口实现；
- [SD SPI 主机驱动](#)：提供 SPI 模式的接口实现；
- 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。

15.1.3 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 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](#)。

示例程序：

- [storage/emmc](#)：访问使用 FAT 文件系统的 eMMC 存储。

15.1.4 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	规格书	AT24C02 driver

常见问题 (FAQ)

- 请参考：[ESP-FAQ 存储部分](#)

15.2 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
存储介质	SPI NOR Flash	SPI NOR Flash、SD/MMC 卡、USB 闪存盘	SPI NOR Flash	SPI NOR Flash、SD/MMC 卡
容量	KB-MB	GB	≤ 128 MB	< 128 MB
目录支持	✗	✓	✗	✓
磨损均衡	✓	可选	✓	✓
读写效率	高	中	中	高
资源占用	低	中	低	极低
掉电保护	✓	✗	✗	✓
加密	✓	✓	✗	✓

备注：

- ✓：支持该特性
- ✗：不支持该特性
- 读写效率和资源占用为相对比较，实际性能取决于具体配置和使用场景。

15.2.1 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。

主要特性：

- **键值对存储**：支持整数、字符串、BLOB 等多种数据类型
- **断电保护**：原子更新，确保数据一致性

- **加密支持**: 支持 AES-XTS 加密
- **磨损均衡**: 内置磨损均衡机制

使用限制:

- **适用场景**: 仅适合配置数据, 不适合频繁写入或大数据
- **分区大小**: 推荐不超过 128 KB
- **空间要求**: 需要足够空间同时存储新旧数据

如需存储较大的 BLOB 或者字符串, 请考虑使用基于磨损均衡库的 FAT 文件系统。

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](#)。

15.2.2 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、USB 闪存盘) 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。

主要特性:

- **广泛兼容性**: 与 PC 和其他平台兼容, 支持标准 FAT 格式
- **多存储介质**: 支持 SPI Flash, SD/MMC 卡和 USB 闪存盘等多种存储介质
- **目录支持**: 支持多级目录结构, 适合存储大量文件
- **加密支持**: 支持分区加密 ([Flash Encryption](#))

使用限制:

- **断电恢复**: 断电恢复能力相对较弱
- **分区大小**: 启用磨损均衡时最小需要 8 个扇区

参考文档:

- [FatFs 与 VFS 配合使用](#)。
- [生成和解析 FATFS 镜像工具](#)。

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。
- [peripherals/usb/host/msc](#): 访问使用 FAT 文件系统的 USB 闪存盘 (需要 ESP32-S2/ESP32-S3)。

15.2.3 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。

主要特性:

- **嵌入式优化**: 专为 NOR Flash 设计, RAM 占用较少
- **静态磨损均衡**: 内置磨损均衡算法
- **断电恢复**: 一定程度的断电后修复功能

使用限制:

- **无目录支持**: 只支持扁平文件结构
- **容量限制**: 最大支持 128 MB Flash
- **性能下降**: 使用率超过 70% 时性能明显下降

- 开发停止：已停止维护

参考文档：

- [SPIFFS 文件系统](#)。
- [生成 SPIFFS 镜像的工具](#)。

示例程序：

- [storage/spiffs](#)：SPIFFS 使用示例。
- [storage/spiffsgen](#)：演示如何使用 SPIFFS 镜像生成工具在构建过程中自动从主机文件夹创建 SPIFFS 镜像。

15.2.4 LittleFS 文件系统

LittleFS 是一个专为微控制器和嵌入式设备设计的基于块的文件系统，原生支持磨损均衡、文件系统一致性检查、断电保护等功能。

主要特性：

- 优异断电恢复：故障安全特性，断电保护能力强
- 动态磨损均衡：自适应磨损均衡算法
- 极低 RAM 占用：固定且极低的 RAM 使用量
- 多存储介质：支持 SPI Flash 和 SD/MMC 卡
- 完整目录支持：支持目录和子目录结构

使用限制：

- 平台兼容性：与其他平台兼容性不如 FAT（主要用于嵌入式）
- 容量建议：建议小于 128 MB 以获得最佳性能
- 第三方维护：需通过 ESP 组件注册表获取
- 文档资源：相比 FAT 文件系统文档资源较少

LittleFS 目前推荐用于一般类型的应用场景，特别是对断电保护要求较高的应用。

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。

15.2.5 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](#)。

15.2.6 存储安全

在选择和使用文件系统时，请注意以下安全相关事项：

- 数据加密：NVS 和 FAT 文件系统支持数据加密，LittleFS 也支持加密功能，SPIFFS 目前不支持加密。
- 掉电保护：NVS 和 LittleFS 具有较好的掉电保护机制，FAT 和 SPIFFS 在掉电时可能存在数据损坏风险。
- 完整性检查：建议定期进行文件系统完整性检查，特别是在生产环境中。

参考文档：

- [存储安全](#)。

15.2.7 文件系统设计建议

- 请参考: [文件系统设计建议](#)。

常见问题 (FAQ)

- 请参考: [ESP-FAQ 存储部分](#)

Chapter 16

电机

16.1 无刷电机

16.1.1 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。

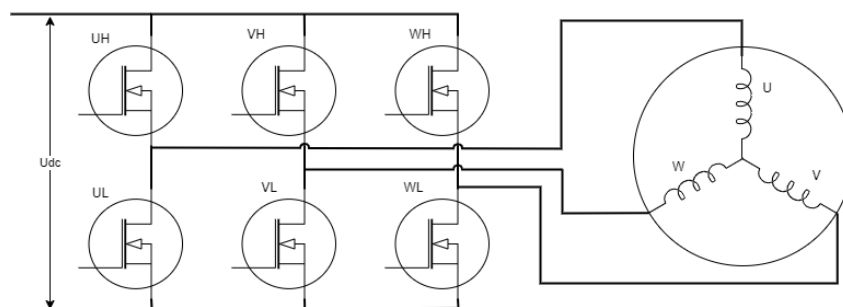


图 2: BLDC 三相逆变电路

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。

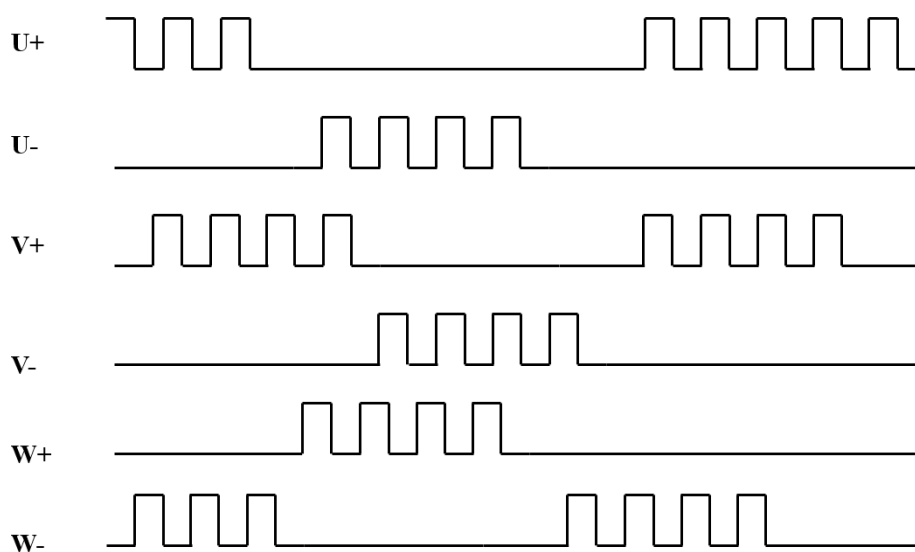


图 3: BLDC PWM 速度控制

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

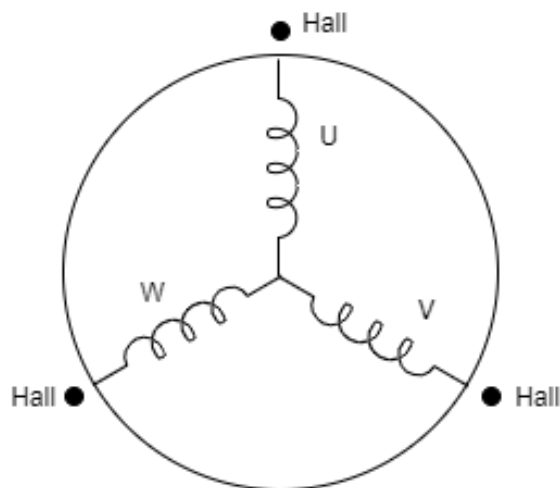


图 4: BLDC 霍尔传感器安装位置

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。

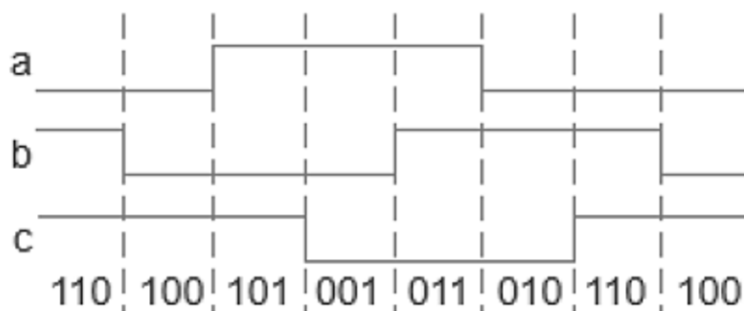


图 5: BLDC 霍尔传感器波形

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° 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）

16.1.2 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， ω 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。

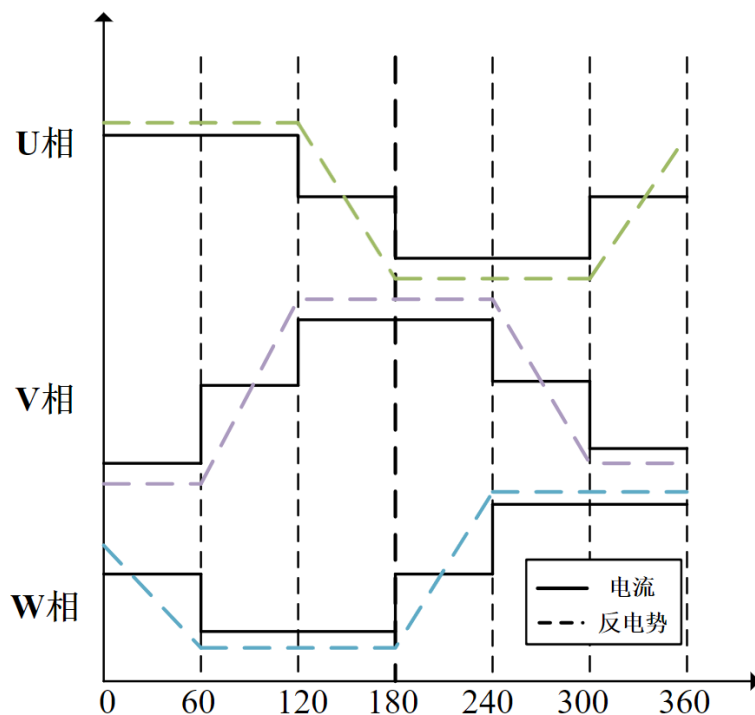


图 6: 电流与反电势波形

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。

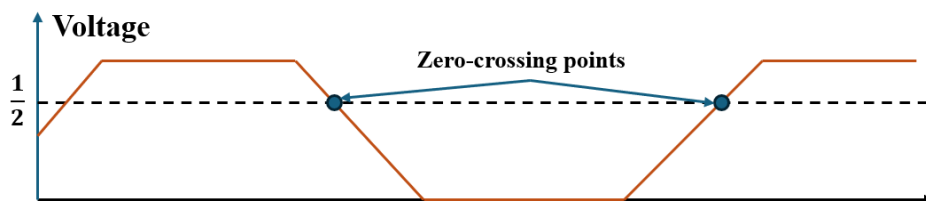


图 7: ADC 过零点检测实现方式

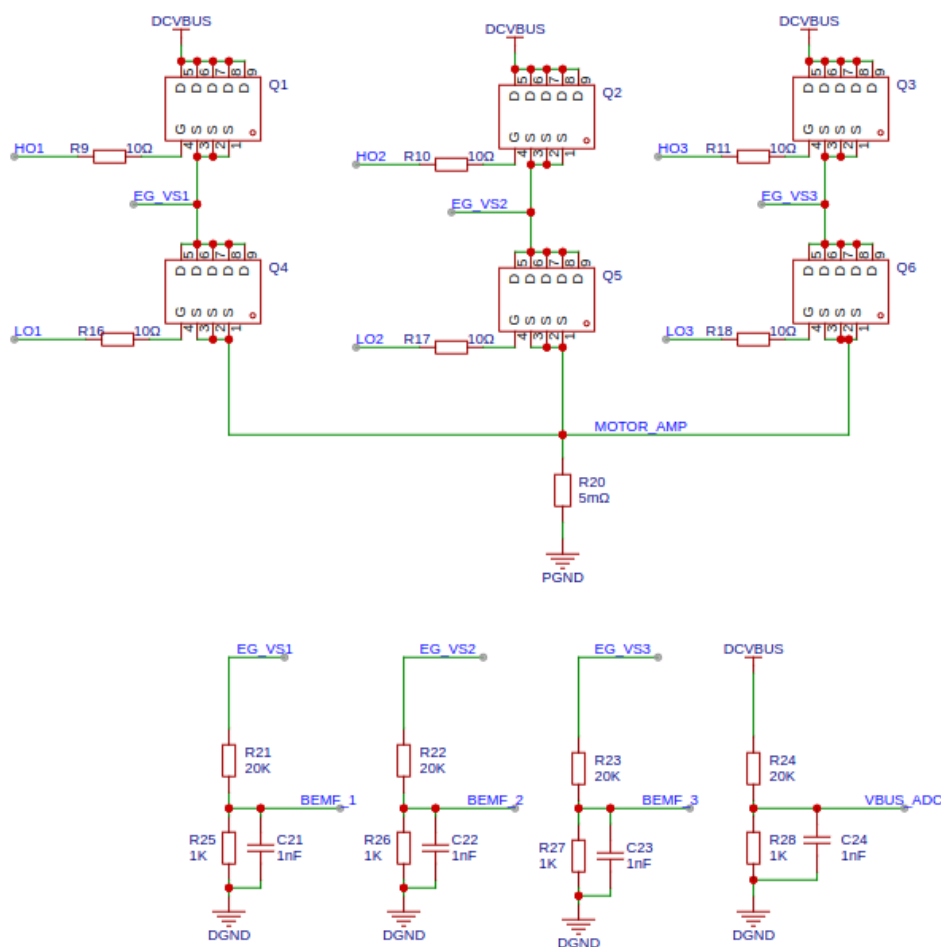


图 8: ADC 过零点检测硬件

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](#)

16.1.3 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， ω 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。

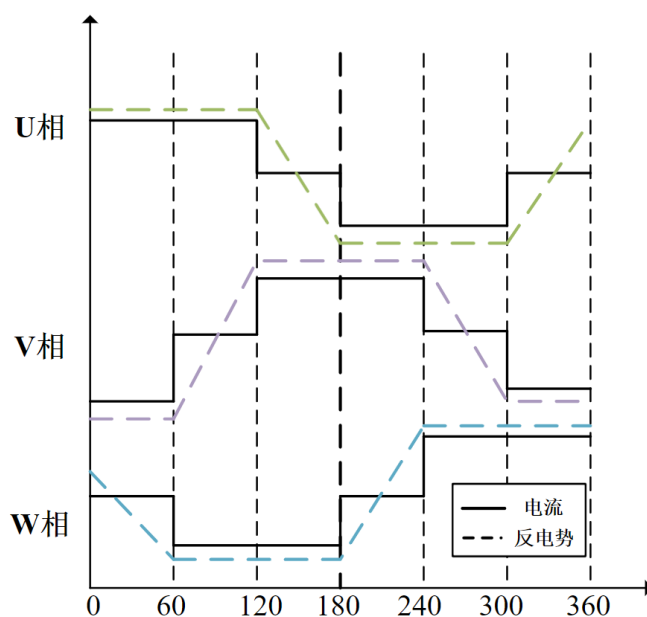


图 9: 电流与反电势波形

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。

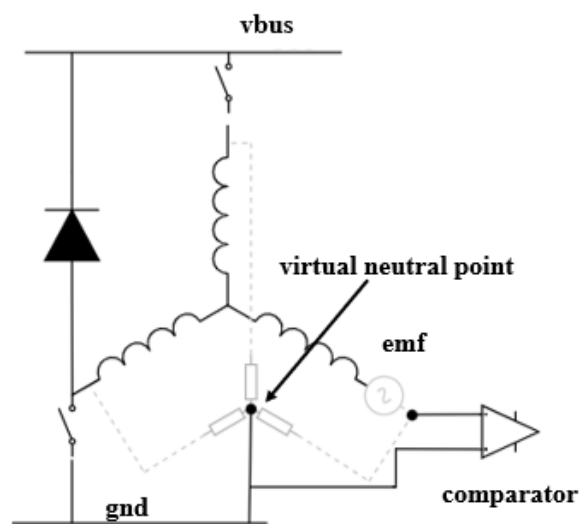


图 10: 比较器过零点原理

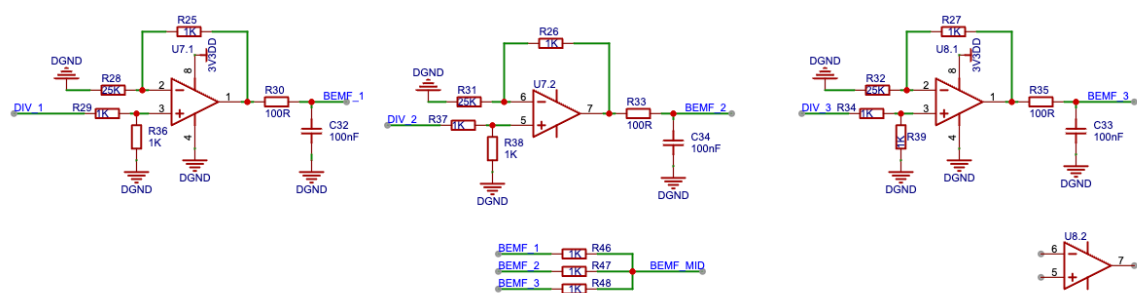


图 11: 比较器过零点硬件

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

16.1.4 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：

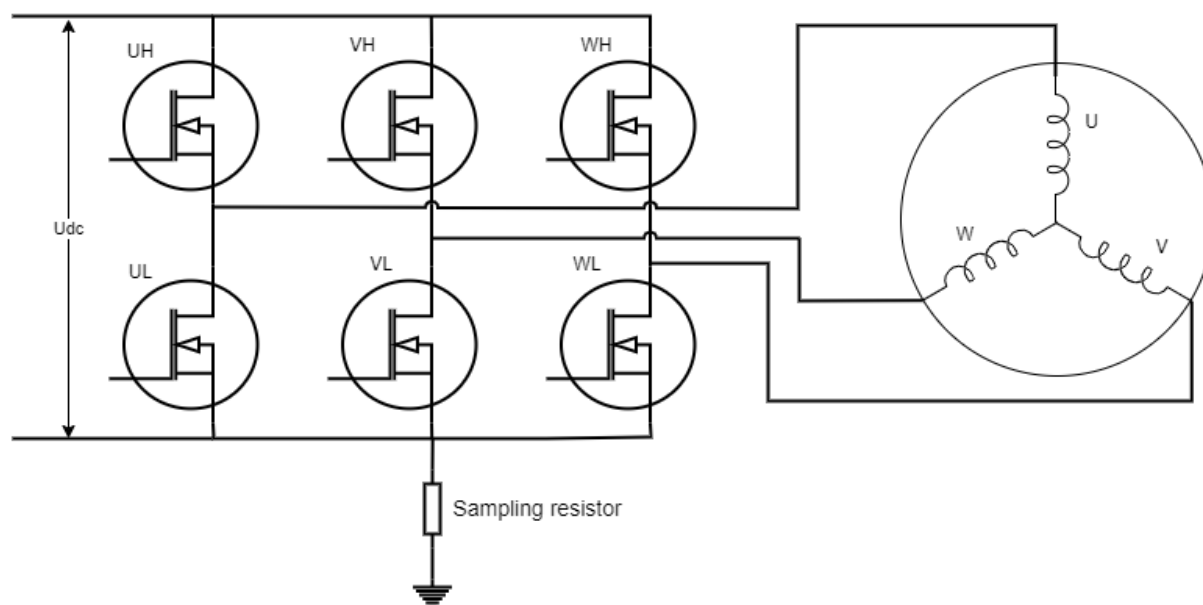


图 12: BLDC 母线电流采集

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

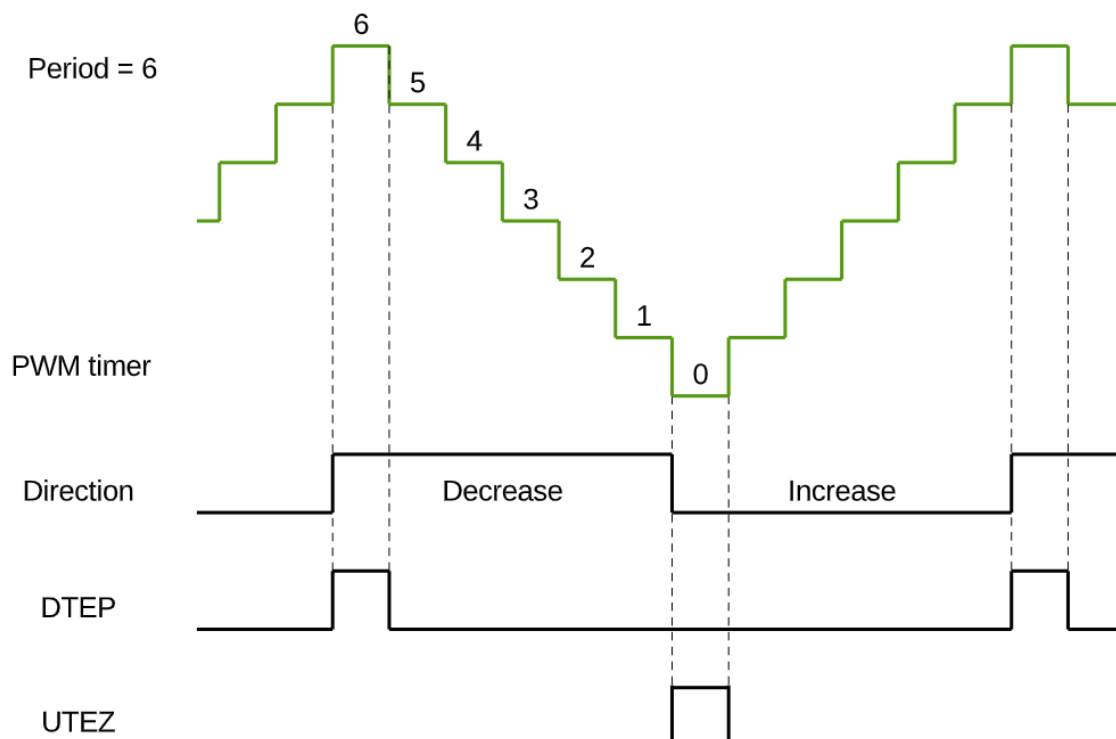


图 13: MCPWM 上升下降模式

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_blcdc_control/include/blcdc_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 blcdc_control_init` (*blcdc_control_handle_t* *handle, *blcdc_control_config_t* *config)

init blcdc control

参数

- **handle** –pointer to blcdc control handle
- **config** –pointer to blcdc 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 blcdc_control_deinit` (*blcdc_control_handle_t* *handle)

deinit blcdc control

参数 **handle** –pointer to blcdc control handle ESP_ERR_INVALID_ARG if handle or config is NULL ESP_OK on success ESP_FAIL on other errors

`esp_err_t blcdc_control_start` (*blcdc_control_handle_t* *handle, uint32_t expect_speed_rpm)

motor start

参数

- **handle** –pointer to blcdc 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 blcdc_control_stop` (*blcdc_control_handle_t* *handle)

motor stop

参数 **handle** –pointer to blcdc control handle

返回 ESP_FAIL if motor stop failed ESP_OK on success

`dir_enum_t blcdc_control_get_dir` (*blcdc_control_handle_t* *handle)

get current motor direction

参数 **handle** –pointer to blcdc control handle

返回 `dir_enum_t` current motor direction

`esp_err_t blcdc_control_set_dir` (*blcdc_control_handle_t* *handle, `dir_enum_t` dir)

set motor direction

参数

- **handle** –pointer to blcdc 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. ≥ 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.

16.2 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。

16.2.1 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。

16.2.2 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);
```

16.2.3 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

16.3 ESP SimpleFOC

esp_simplefoc 是一个基于 [Arduino-FOC](#) 的 FOC 组件，针对支持 LEDC 与 MCPWM 功能的 ESP 芯片开发。支持以下特性：

- 支持电压控制
- 支持通过 [SimpleFOCStudio](#) 来调整和配置电机控制参数
- 兼容 [Arduino-FOC](#) 控制例程
- 采用 [IQMath](#) 实现 FOC 运算加速
- 支持手动选择 LEDC 或 MCPWM 外设来控制 MOS 驱动器，最高可实现四路无刷电机控制

FOC (Field-Oriented Control) 是一种用于无刷直流电机磁场定向控制算法，能够产生平滑的扭矩、速度与位置控制。有关更友好的解释，请查看 [Arduino-FOC 官方文档](#)。

16.3.1 在您的项目中使用 SimpleFOC

1. 使用 `idf.py add_dependency` 命令将组件添加到项目中

```
idf.py add-dependency "espressif/esp_simplefoc"
```

2. 使用 `idf.py menuconfig` 配置 FreeRTOS (Top) → Component config → FreeRTOS → Kernel

- `configTICK_RATE_HZ`: FreeRTOS 的时钟 Tick 的频率，默认情况下为 100，在 FOC 场景下需设置为 1000

3. 配置电机参数、MOS 驱动器控制参数与引脚、角度传感器、控制器

电机参数

- `pp`: 极对数，您需要根据实际的三相无刷电机参数设置对应的电机极对数
- `R`: 电机相电阻，默认为 `NOT_SET`
- `KV`: 电机 KV 值，默认为 `NOT_SET`
- `L`: 电机相电感，默认为 `NOT_SET`

对于极对数为 14 的三相无刷电机而言，对应的电机实例化为：

```
BLDCMotor motor = BLDCMotor(14);
```

MOS 驱动器控制参数与引脚

- `pwm pin`: PWM 输出引脚，您需要根据实际的 MOS 驱动器电路来设置对应的 PWM 输出引脚
- `enable pin`: MOS 驱动器使能脚，若您的 MOS 驱动器需要手动使能，请手动填入对应的 GPIO，默认为 `NOT_SET`
- `voltage_power_supply`: MOS 驱动器的供电电压
- `voltage_limit`: MOS 驱动器的电压限制

对于 3PWM 模式的 12V MOS 驱动器而言，对应的驱动实例化与参数为：

```
BLDCDriver3PWM driver = BLDCDriver3PWM(4, 5, 6);
driver.voltage_power_supply = 12;
driver.voltage_limit = 11;
driver.init({1, 2, 3});
motor.linkDriver(&driver);
```

角度传感器

您需要根据实际使用的角度传感器进行对应的实例化，当前支持的角度传感器：

- `as5048a`: 支持 SPI 方式读取角度数据
- `mt6701`: 支持 I2C、SPI 方式读取角度数据
- `as5600`: 支持 I2C 方式读取角度数据

对于 AS5600 角度传感器实例化与对应的绑定操作为：

```
AS5600 as5600 = AS5600(I2C_NUM_0, GPIO_NUM_1, GPIO_NUM_2);
as5600.init();
motor.linkSensor(&as5600);
```

控制器

`esp_simplefoc` 支持多种控制方式，您需要根据实际的应用需求进行选择：

- `torque`: 力矩控制
- `velocity`: 速度控制
- `angle`: 角度控制
- `velocity_openloop`: 速度开环控制
- `angle_openloop`: 角度开环控制

备注：若在硬件验证阶段，可优先选择开环控制方案，以实现快速验证。

4. FOC 初始化与循环控制

在完成上述参数设置后，您需要运行电机 `init` 与 `initFOC`，以实现电机初始化校准，并启动 `loopFOC`：

```
motor.init();
motor.initFOC();
while (1) {
    motor.loopFOC();
    motor.move(target_value);
    command.run();
}
```

备注：若 `initFOC` 所估计的极对数与实际填入的极对数不符合，请排查电机极对数，并尽可能缩减磁环与角度传感器之间的间距。

5. 运行 `idf.py build flash` 进行初始下载, 之后可观察电机的实际运行效果

16.3.2 API 参考

`esp_simplefoc` 的 API 接口与 [Arduino-FOC](#), 可参考 [Arduino-FOC API 文档](#)

Chapter 17

安全 & 加密

17.1 Flash 加密

Flash 加密是 ESP32 系列芯片的重要安全特性，用于保护存储在外部 Flash 中固件和数据的机密性。启用 Flash 加密后，Flash 中的内容将被透明加密，防止通过物理方式读取获得明文固件和数据。

Flash 加密的核心特点：

- **透明加密解密**：通过 MMU 缓存访问时自动处理加密解密
- **硬件实现**：加密密钥存储在 eFuse 中，软件无法直接访问
- **XTS-AES 算法**：采用专为存储设备设计的安全加密算法
- **分区选择性加密**：可指定需要加密的分区类型

更多详细信息请参考：[ESP-IDF Flash 加密文档](#)

17.1.1 Flash 加密与安全启动

Flash 加密与安全启动是互补的安全功能：

- **Flash 加密**：保护数据机密性，防止读取 Flash 内容
- **安全启动**：保证代码完整性，防止运行未授权固件

建议在生产环境中同时启用两项功能以获得最佳安全保护。详情参考：[Flash 加密与安全启动](#)

17.1.2 加密范围

Flash 加密会自动加密以下内容：

- 二级引导加载程序
- 所有 app 类型分区
- 分区表
- NVS 密钥分区
- 用户标记为“encrypted”的分区

重要提示：NVS 分区加密和 Flash 加密是两个独立的功能。需要使用 NVS 加密功能。详见：[NVS 加密文档](#)

加密范围详细说明：[加密分区](#)

17.1.3 Flash 加密功能支持列表

以下是 ESP32 系列芯片 Flash 加密功能对比表，包含加密算法、密钥长度/类型、硬件支持、密钥存储方式、密钥来源（精简描述）：

表 1: ESP32 系列芯片 Flash 加密功能对比

芯片型号	支持的加密算法	密钥长度/类型	硬件加速/支持	密钥存储方式	密钥来源
ESP32	AES-256 (自定义实现)	256-bit	有	eFuse	硬件随机数生成
ESP32-S2	XTS-AES-128、XTS-AES-256	256-bit 或 512-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成
ESP32-S3	XTS-AES-128、XTS-AES-256	256-bit 或 512-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成
ESP32-C2	XTS-AES-128	256-bit	有	eFuse (BLOCK_KEY0)	硬件随机数或主机生成
ESP32-C3	XTS-AES-128	256-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成
ESP32-C5	XTS-AES-128	256-bit	有	eFuse (BLOCK_KEYN) 或 Key Manager	硬件随机数、主机生成或 Key Manager
ESP32-C6	XTS-AES-128	256-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成
ESP32-H2	XTS-AES-128	256-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成
ESP32-P4	XTS-AES-128、XTS-AES-256	256-bit 或 512-bit	有	eFuse (BLOCK_KEYN)	硬件随机数或主机生成

说明： - ESP32-C5 支持 Key Manager 作为密钥来源和存储方式（参考：[ESP32-C5 SoC 功能宏定义](#)）。 - 其它芯片均为 eFuse 存储，密钥可由硬件随机数生成或主机生成后烧录（参考：[ESP32-S2 Flash 加密](#) 和 [ESP32-C3 Flash 加密](#)）。

17.1.4 开发模式和发布模式

Flash 加密提供两种模式：

开发模式 (Development Mode)

- 允许多次烧录明文固件
- 引导加载程序自动处理加密
- 适用于开发和测试阶段
- 安全性相对较低

发布模式 (Release Mode)

- 禁用明文固件烧录
- 只能通过 OTA 更新明文固件
- 适用于生产环境
- 提供最高安全性

状态检查：

- 使用 `esp_flash_encryption_enabled()` 检查 Flash 加密状态
- 使用 `esp_get_flash_encryption_mode()` 获取加密模式（开发模式或发布模式）

配置方法详见：[Flash 加密设置](#)

17.1.5 加密过程

Flash 加密的基本过程：

1. **首次启动**: 检测 eFuse 状态, 启动加密流程
2. **密钥生成**: 生成随机加密密钥并存储到 eFuse (可跳过, 支持烧录自定义密钥)
3. **就地加密**: 加密 Flash 中的指定分区内容
4. **设置标志**: 标记 Flash 加密已启用
5. **重启系统**: 开始正常的加密模式运行

完整的加密过程说明: [Flash 的加密过程](#)

17.1.6 应用程序访问加密分区

应用程序可以透明地访问加密的 Flash 内容:

读取加密 Flash:

- 使用 `esp_partition_read()` 或 `esp_flash_read_encrypted()` 读取明文内容 (自动解密)
- 使用 `esp_flash_read()` 读取原始加密数据 (不解密)

写入加密 Flash:

- 使用 `esp_partition_write()` 写入明文内容 (自动加密)
- 使用 `esp_flash_write_encrypted()` 写入原始加密内容 (跳过自动加密)

更多 API 详情: [在加密的 Flash 中读写数据](#)

17.1.7 重新烧录 Flash

开发模式下:

- 通过 `idf.py encrypted-app-flash` 烧录新的应用程序明文, 烧录时会自动加密
- 通过 `idf.py encrypted-flash` 烧录所有分区明文, 烧录时会自动加密

发布模式下:

- 只能通过 OTA 更新明文固件
- 只能手动烧录密文固件 (仅当 UART ROM Downloads 启用时支持)

详细说明: [重新烧录更新后的分区](#)

17.1.8 取消加密

仅开发模式支持: 在加密状态下通过烧录 eFuse `SPI_BOOT_CRYPT_CNT` 以禁用 Flash 加密 (发布模式无法取消)。

警告: 每个芯片只有有限次数的开关加密次数, 通常为 3 次 (关闭->开启->关闭->开启), 请谨慎操作。

详细步骤: [关闭 Flash 加密](#)

17.1.9 示例代码

完整的 Flash 加密使用示例请参考:

- [ESP-IDF Flash 加密示例](#)
- [安全功能综合示例](#)

这些示例展示了:

- Flash 加密状态检查
- 加密分区读写操作
- NVS 和 FATFS 在加密环境下的使用
- 开发模式和发布模式的配置方法

17.1.10 最佳实践

1. 生产环境使用发布模式
2. 每个设备使用唯一密钥
3. 结合安全启动使用
4. 合理规划分区加密策略
5. 测试 OTA 更新流程

更多最佳实践：[Flash 加密最佳实践](#)

17.1.11 常见问题 (FAQ)

- 请参考：[ESP-FAQ 安全部分](#)

17.2 安全启动

安全启动是 ESP32 系列芯片的关键安全防护机制，通过在启动过程和 OTA 更新时对软件实施数字签名验证，确保设备只能执行经过授权的固件。该机制建立了从引导加载程序到应用程序的完整信任链，为设备提供完备的代码完整性保护。

安全启动的核心特点：

- **代码完整性保护**：通过密码学验证确保固件代码未被篡改或替换，防止运行未经授权的恶意程序
- **完整信任链保护**：建立从引导加载程序到应用程序的端到端验证机制，确保启动流程中的每个环节都经过验证
- **支持多种签名算法**：可选择 RSA-PSS 或 ECDSA 作为签名算法，并通过硬件加速确保高效验证
- **硬件安全存储**：将签名密钥摘要存储在受保护的 eFuse 中，提供严格的硬件级读写保护机制

更多详细信息请参考：[ESP-IDF 安全启动 v2 文档](#)

17.2.1 安全启动与 Flash 加密

安全启动与 Flash 加密是互补的安全功能：

- **安全启动**：保证代码完整性，防止运行未授权固件
- **Flash 加密**：保护数据机密性，防止读取 Flash 代码和数据

建议在生产环境中同时启用两项功能以获得最佳安全保护。详情参考：[安全启动与 Flash 加密](#)

17.2.2 安全启动信任链

安全启动会验证以下软件组件：

- 一级引导加载程序（ROM 代码）作为芯片固化代码，无法被更改或破坏，作为信任链的根基
- 一级引导加载程序通过数字签名验证二级引导加载程序（bootloader）的完整性和真实性
- 二级引导加载程序通过数字签名验证应用程序（app）二进制文件的完整性和真实性
- 应用程序在 OTA 升级过程中验证新固件的数字签名，确保更新的安全性

详细验证过程：[安全启动 v2 过程](#)

17.2.3 安全启动功能支持列表

表 2: 安全启动功能支持列表

芯片型号	支持算法 (曲线/位数)	签名验证时间 (CPU 频率)	最大公钥数量	备注/特殊说明
ESP32 (ECO V3+)	RSA-3072 (RSA-PSS, Secure Boot V2) AES (V1)	未明确给出	1	仅支持 1 把公钥, Secure Boot V2 需 ECO V3 及以上
ESP32-S2	RSA-3072 (RSA-PSS)	未明确给出	3	最多 3 把公钥, 可吊销
ESP32-S3	RSA-3072 (RSA-PSS)	未明确给出	3	最多 3 把公钥, 可吊销
ESP32-C2	ECDSA-192 (NISTP192) ECDSA-256 (NISTP256, 推荐)	未明确给出	1	仅支持 ECDSA, 且仅 1 把公钥 ESP32-C2 Kconfig Reference
ESP32-C3 (v0.3 及以上)	RSA-3072 (RSA-PSS)	未明确给出	3	最多 3 把公钥, 可吊销
ESP32-C5	RSA-3072 ECDSA-256 (P-256) ECDSA-384 (P-384)	RSA-3072: 12.1 ms ECDSA-256: 5.6 ms ECDSA-384: 20.6 ms (48 MHz)	3	仅可选一种算法 ESP32-C5 Secure Boot v2
ESP32-C6	RSA-3072 ECDSA-256 (P-256)	RSA-3072: 10.2 ms ECDSA-256: 83.9 ms (40 MHz)	3	仅可选一种算法 ESP32-C6 Secure Boot v2
ESP32-H2	RSA-3072 ECDSA-256 (P-256)	RSA-3072: 18.3 ms ECDSA-256: 76.2 ms (32 MHz)	3	仅可选一种算法 ESP32-H2 Secure Boot v2
ESP32-P4	RSA-3072 ECDSA-256 (P-256)	RSA-3072: 14.8 ms ECDSA-256: 61.1 ms (40 MHz)	3	仅可选一种算法 ESP32-P4 Secure Boot v2

说明: - “最大公钥数量”指可在 eFuse 中存储的公钥数量, 部分芯片支持密钥吊销。 - “签名验证时间”指 ROM 引导加载器验证 bootloader 镜像签名所需时间, 实际启动时间可能略有不同。 - 若需更详细的配置和使用建议, 请查阅对应芯片的官方文档。

17.2.4 签名与密钥管理

在信息安全中, 数字签名 (Digital Signature) 是一种基于非对称加密的完整性验证机制。它的核心目的是: 证明数据的来源真实性 (Authenticity), 以及确保数据在传输或存储过程中未被篡改 (Integrity)。

签名过程使用的是一对密钥:

- 私钥 (Private Key): 用于对数据进行签名, 私钥必须严格保密, 保存在安全的环境中 (本地固件开发环境或远程签名服务器)。
- 公钥 (Public Key): 用于验证签名的合法性, 公钥可以安全地分发给任何需要验证签名的实体。

安全启动 v2 支持多种非对称加密算法, 具体取决于芯片型号:

- **RSA-PSS:** RSA-3072 密钥, 适用于快速启动的场景
- **ECDSA:** 支持 P-256 和 P-192 曲线, 密钥长度更短但验证时间较长 (受限于当前 ECDSA 硬件加速能力)

17.2.5 签名块格式

签名块 (Signature Block) 是 ESP32-S3 Secure Boot v2 中用于验证镜像完整性和来源可信性的关键结构。它是附加在 bootloader 或应用程序镜像末尾的一个固定格式数据块, 包含签名信息、公钥以及辅助验证字段。

- 签名块主要包含镜像的 SHA-256 哈希值、公钥、签名结果等
- 签名块本身占据 1216 字节，位于镜像末尾的 4KB 对齐边界，且具有 CRC32 校验
- 每个镜像可附加最多 3 个签名块，用于支持多密钥验证与撤销机制

签名块详细定义：[签名块格式](#)

17.2.6 启用硬件安全启动

方法一：通过配置项启用 (menuconfig)：

在 ESP-IDF 中启用 Secure Boot v2 最简单的方式是通过 menuconfig 配置项。在构建 bootloader 时，构建系统会在编译 bootloader 和 app 镜像时自动插入签名流程。首次启动时，bootloader 会自动烧写 eFuse 以启用 Secure Boot，并撤销未使用的密钥槽。这种方式适合开发阶段，流程自动化、集成度高，但由于密钥撤销不可逆，量产时需谨慎使用。

1. 打开项目配置菜单 Security features 启用 Enable hardware Secure Boot in bootloader
2. 选择 Secure Boot v2 签名方案，根据不同芯片，可选的签名算法为“RSA”或 ECDSA
3. 指定签名密钥文件路径，密钥文件（例如 RSA 3072）可通过 idf.py secure-generate-signing-key 或 openssl genrsa 指令生成
4. 使用 idf.py build 指令将直接编译、对齐填充、生成附带签名块的 app 镜像
5. 使用 idf.py bootloader 指令生成启用了安全启动且已经附带签名块的 bootloader 镜像
6. 为了安全起见，默认情况下启用了安全启动的 bootloader 需要使用 esptool.py write_flash 指令单独烧写，idf.py flash 只能烧写应用程序和分区表等分区
7. 重启设备，bootloader 会在首次运行时自动烧写 eFuse 以启用 Secure Boot，并将计算出签名块的公钥摘要写入 eFuse，并撤销未使用的密钥槽
8. 之后的启动过程中，eFuse 中的公钥摘要将用于验证 bootloader 和 app 镜像的签名，确保设备仅运行对应私钥签名后的固件

注意事项：启用安全启动后，bootloader 镜像大小将增加，可能需要重新调整分区表大小。私钥必须妥善保管，如果私钥泄露，攻击者可以生成有效签名的恶意固件，如果密钥丢失，设备将无法更新固件。

方法二：外部工具启用：

另一种方式是使用 espefuse.py 工具在烧录前手动配置 eFuse，包括写入公钥摘要、设置密钥用途、启用 Secure Boot 标志位等。这种方式不会触发 bootloader 的自动密钥撤销逻辑，因此可以保留未使用的密钥槽，便于未来密钥轮换或 OTA 签名更新。它适合量产环境，安全性高、控制精细，但操作复杂，需要严格的密钥管理和烧录流程。此外，你可以使用 [远程签名](#) 或 [外部 HSM](#) 来生成签名块，进一步提升私钥安全性

1. 生成私钥（可在本地或远程环境中生成）：创建用于签名固件的私钥
2. 生成公钥摘要：对公钥进行 SHA-256 哈希计算
3. 烧录摘要：将公钥摘要写入 eFuse 特定区域
4. 启用安全启动：置位相关 eFuse 标志位
5. 撤销未用密钥槽：防止未授权密钥被添加
6. 烧录安全配置：设置其他安全相关的 eFuse 位
7. 配置项目：禁用自动签名以使用外部签名工具
8. 签名镜像：对 bootloader 和 app 进行签名
9. 烧录镜像：将签名后的固件写入设备
10. 启用安全下载：最终锁定所有安全配置

详细配置方法和指令详见：[外部启用 v2](#)

17.2.7 密钥撤销

对于已烧录多个公钥的芯片，可以通过撤销密钥防止已泄露或不再使用的密钥被利用。撤销密钥后，使用该密钥签名的固件将无法通过验证，从而保护设备免受潜在攻击。

实施前提：

密钥撤销功能需要满足以下前提条件：

1. 芯片必须支持多个公钥 (>1) 且具备密钥撤销功能
2. 设备出厂时已在 eFuse 中烧录至少两个公钥摘要 (如 key #0 和 key #1)
3. 二级引导加载程序 (bootloader) 使用多重签名机制, 已由 key #0 和 key #1 对应的私钥进行签名
4. 应用程序 (app) 仅使用其中一个密钥 (如 key #0) 进行签名

更多关于多重签名的信息, 请参考: [多重签名](#)

实施步骤:

1. 使用新密钥 (如 key #1) 签名的 OTA 固件通过当前应用验证后写入备用分区
2. 新应用启动后验证二级引导加载程序 (bootloader) 签名正常 (确认 bootloader key #1 可用)
3. 调用 `esp_ota_revoke_secure_boot_public_key ()` 撤销旧密钥 (key #0 撤销)
4. 旧密钥 (key #0) 被撤销后, 使用该密钥签名的固件将无法通过验证
5. 二级引导加载程序 (bootloader) 和应用程序 (app) 均使用新密钥 (key #1) 签名验证

密钥撤销详情: [密钥撤销](#)

17.2.8 启用后的限制

安全限制:

- 启用硬件安全启动后, 无禁用方法
- 更新的引导加载程序或应用程序必须使用匹配的密钥签名
- 禁用 USB-OTG USB 栈, 不允许通过串行仿真或 DFU 更新
- 禁用进一步的 eFuse 读保护, 防止攻击者读保护安全启动公钥摘要

调试接口:

- JTAG 接口默认被禁用
- UART 下载模式切换到安全模式

详细限制说明: [安全启动启用后的限制](#)

17.2.9 软件签名验证

软件签名验证提供了一种轻量级的签名验证机制, 和硬件安全启动使用相同的签名方案, 但仅在 OTA 更新时验证新镜像, 适用于对启动速度敏感或物理安全要求较低的场景, 但不具备完整的启动链保护能力。

软件签名 vs 硬件安全启动:

表 3: 软件签名 vs 硬件安全启动

项目	☑ Secure Boot v2 (硬件)	☑ 软件签名验证机制 (无 Secure Boot)
启动时验证	☑ 验证 bootloader 和 app 镜像签名	✗ 不验证当前 app, 假定其可信
OTA 更新验证	☑ 使用 eFuse 中密钥验证新 app 签名	☑ 使用当前 app 的签名块公钥验证新 app
安全根 (Root of Trust)	eFuse 中烧录的公钥摘要	当前运行 app 的签名块中的公钥
eFuse 配置要求	必须烧录 SECURE_BOOT_EN 和 KEY_DIGESTx	无需烧录 Secure Boot 相关 eFuse
密钥撤销机制	☑ 支持 KEY_REVOKE_x 和激进撤销策略	✗ 不支持密钥撤销
签名块支持数量	☑ 最多 3 个签名块, 支持多密钥验证	✗ 仅使用第一个签名块, 忽略其他
防篡改能力	☑ 防止 Flash 被替换或注入恶意代码	✗ 无法防止物理攻击或 Flash 替换
启动性能	✗ 启动时有签名验证开销	☑ 启动更快, 无验证延迟
开发便利性	✗ 启用 eFuse 后不可逆, 需谨慎操作	☑ 无不可逆操作, 适合开发调试
适用场景	量产部署、高安全性要求设备	开发测试、启动速度敏感、物理安全可控环境
密钥管理灵活性	☑ 支持多密钥轮换与撤销	✗ 仅依赖当前 app 的公钥, 无法轮换
推荐使用	☑ 官方推荐用于正式产品	△ 仅在明确威胁模型下使用, 需谨慎评估

配置方法：

启用 CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT 选项。

详情参考：[无硬件安全启动的签名应用程序验证](#)

17.2.10 示例代码

完整的安全启动使用示例请参考：

- [ESP-IDF 安全启动示例](#)
- [安全功能综合示例](#)

这些示例展示了：

- 安全启动状态检查
- 签名密钥生成和管理
- 多密钥签名和撤销
- 与 Flash 加密的配合使用

17.2.11 最佳实践

1. 使用高质量熵源生成签名密钥
2. 始终保持签名密钥私密
3. 避免第三方观察密钥生成或签名过程
4. 启用所有安全启动配置选项
5. 结合 Flash 加密使用
6. 使用多个密钥减少单点故障
7. 制定密钥轮换策略

更多最佳实践：[安全启动最佳实践](#)

17.2.12 常见问题 (FAQ)

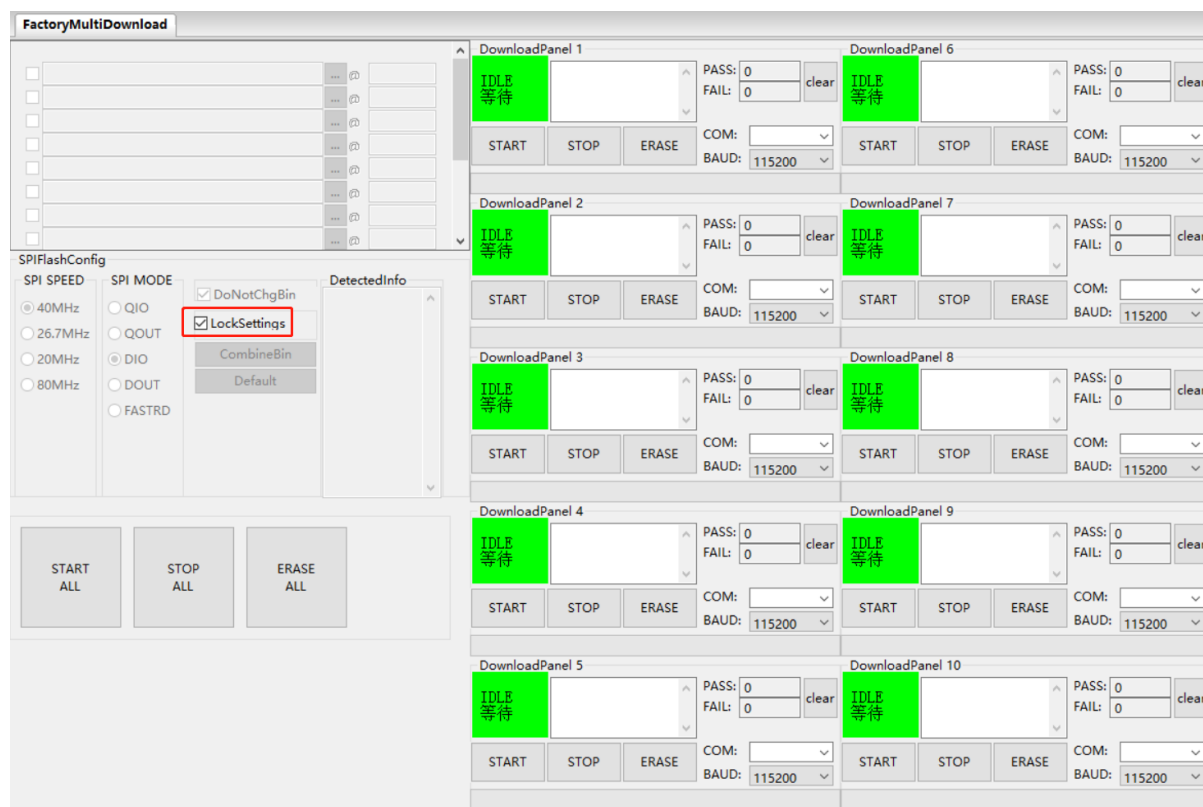
- 请参考：[ESP-FAQ 安全部分](#)

17.3 量产方案

本文档集合了用于乐鑫芯片量产的多种工具，包括烧录、测试、治具规范、配网及射频工具等内容。

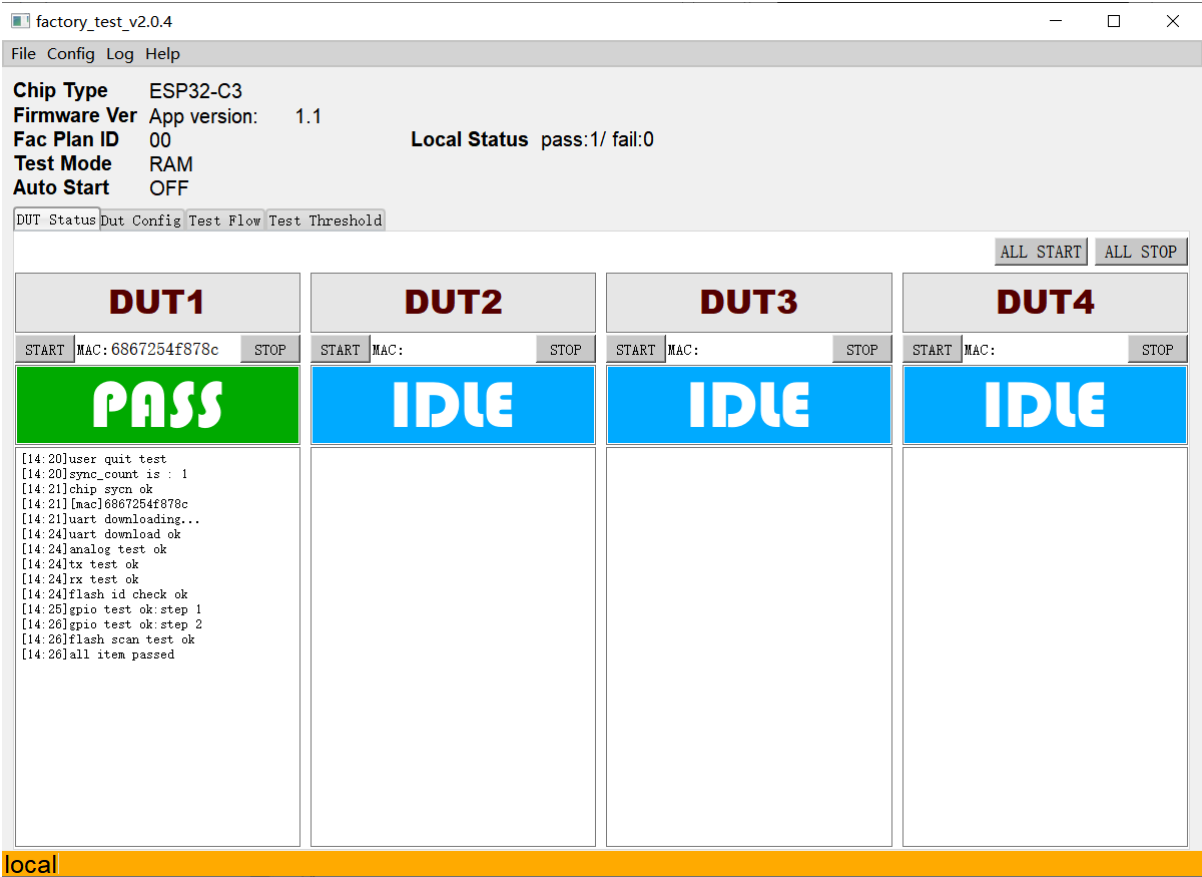
17.3.1 Flash 下载工具

该工具面向生产环节，用于将量产固件高效烧录到芯片 Flash，支持多芯片并行烧录和多种烧录参数配置。它解决了量产过程中固件更新、批量烧录、自动化测试等问题，提升生产效率，降低人为失误。适用于产线固件部署、批量升级、调试流程。参考 [ESP32 生产阶段](#)。



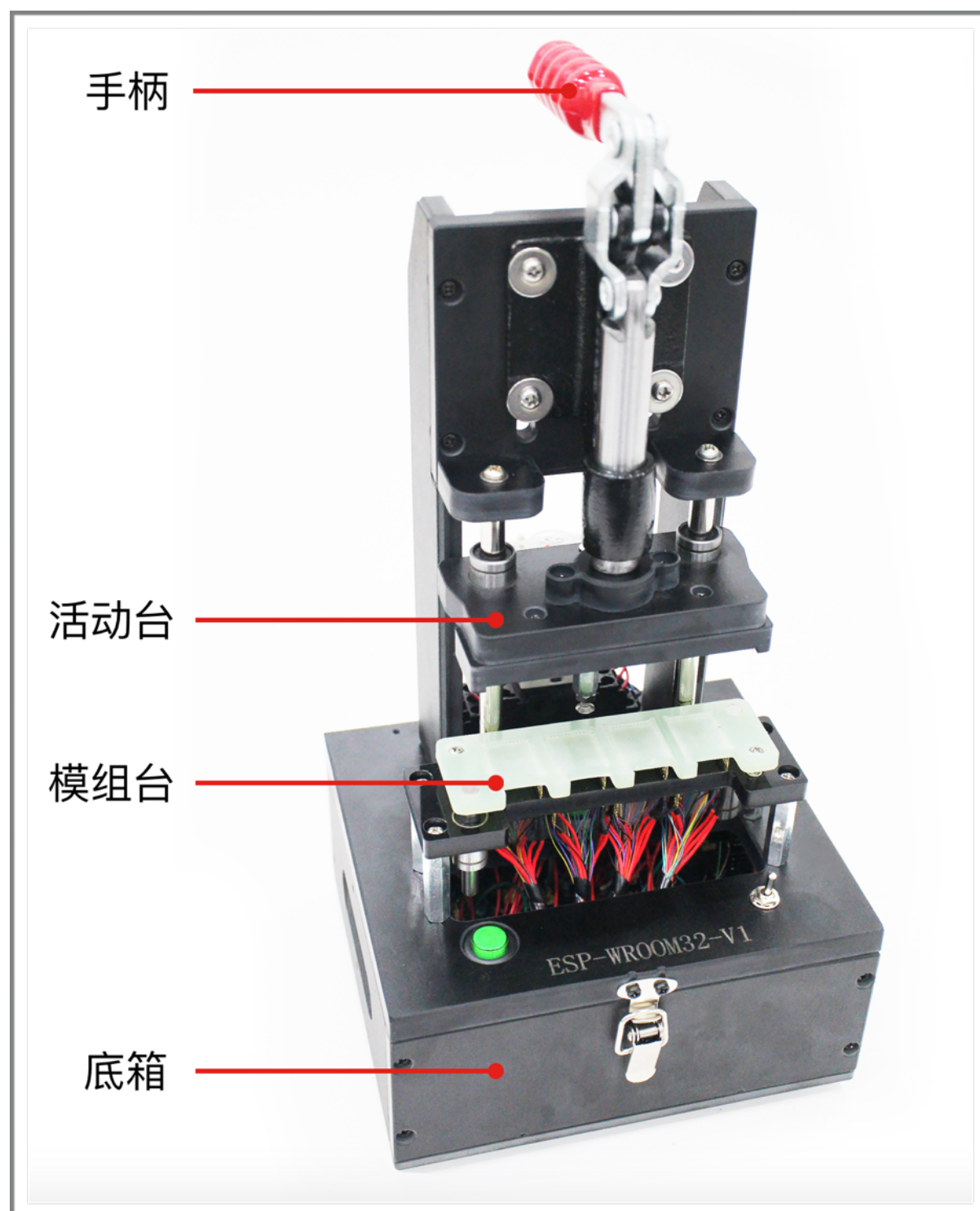
17.3.2 乐鑫产测工具

该工具是一拖四的产线测试上位机，采用”RAM 运行测试”方式，支持自动化配置和执行测试流程。功能包括 RF 性能测试、GPIO 导通测试、固件版本校验、Flash 读写/ID 测试等，能够自动生成测试日志和统计报表，并支持测试 bin、阈值和系统配置管理。它解决了产线测试效率低、人工操作繁琐、数据统计难等问题，适用于大批量生产测试、产线质量管控。参考 [ESP32 产测指南-产测工具](#)。



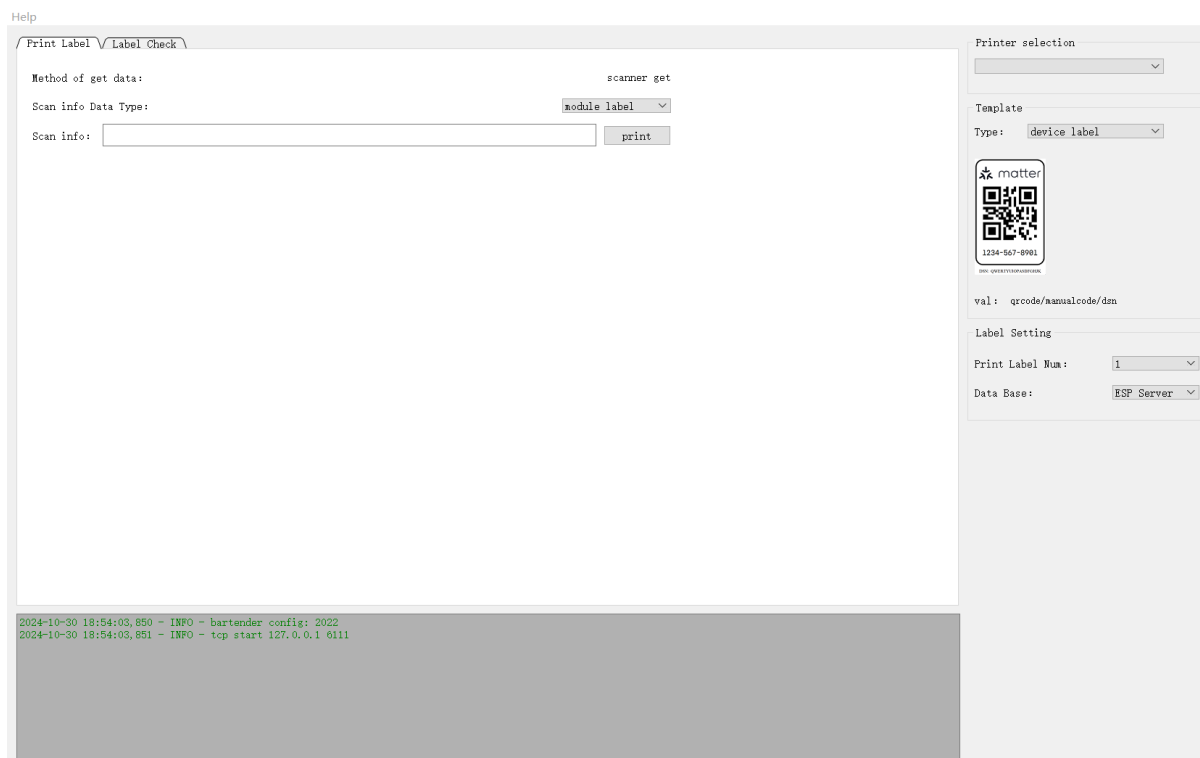
17.3.3 模组治具制作规范

该规范指导产测治具的结构设计与制作工艺，确保测试夹具在量产过程中具备一致性和可靠性。通过标准化治具设计，减少因治具差异导致的测试误差和不良品率，提升测试效率和产品质量。适用于量产测试夹具开发、产线治具选型。参考 [ESP32 生产阶段](#)。



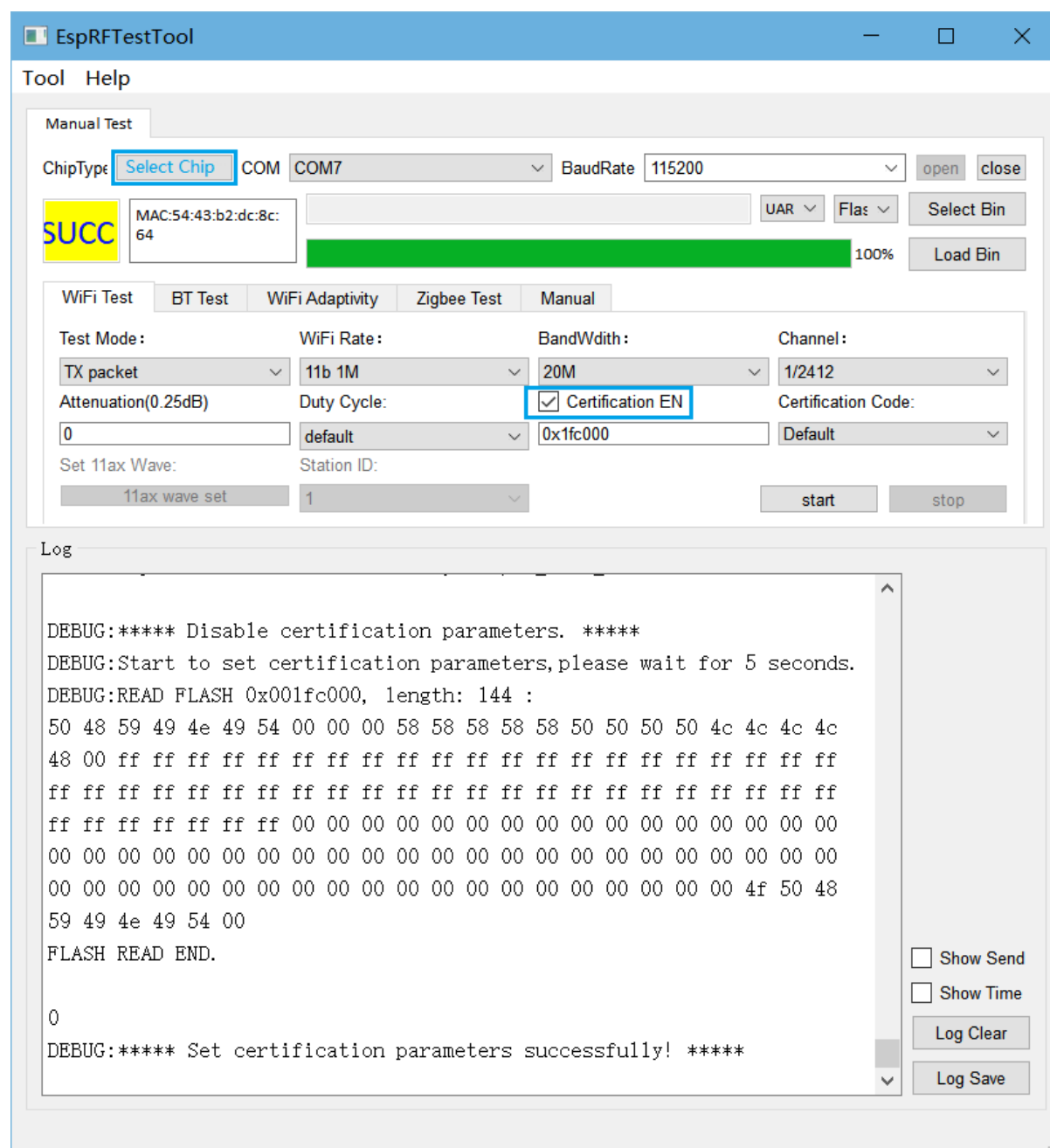
17.3.4 Matter QR 二维码生成工具

该工具用于生成 Matter 设备的配网二维码，支持批量生成和自定义参数配置。它解决了产线设备快速入网、自动化配网、批量验证等问题，提升量产设备的联网效率和一致性。适用于 Matter 设备生产、产线配网、入网验证等场景。参考 [ESP32 生产阶段](#)。



17.3.5 EspRFTestTool 工具包

- **EspRFTestTool**: 用于射频测试的上位机工具，支持 Wi-Fi 非信令测试、蓝牙/低功耗蓝牙非信令测试、Wi-Fi 自适应测试等。该工具集成串口配置、固件快速烧录、测试参数配置与状态日志窗口，能够帮助研发人员在芯片开发阶段快速验证关键 RF 指标，发现并定位射频性能问题，提升产品可靠性。适用于芯片初期调试、射频性能评估、认证前预检等场景。下载入口见 [EspRFTestTool 工具包](#)。
- **DownloadTool**: 用于将测试固件烧录到芯片，支持 RAM 和 Flash 两种烧录模式，并可指定烧录地址。该工具主要解决射频测试前固件快速部署的问题，确保测试环境一致性，减少人工操作失误。适用于 RF 测试准备、固件版本切换、批量烧录等需求。参考 [ESP32 工具包](#)。
- **PowerLimitTool**: 用于生成定制化的 phy_init_data 固件，支持灵活配置射频功率限制等参数。该工具帮助用户满足不同国家/地区的射频法规要求，提升产品合规性，并可配合 RF 测试与认证流程使用。适用于认证准备、特殊应用场景功率调整等。参考 [ESP32 工具包](#)。



Chapter 18

电工 & 照明方案

18.1 球泡灯驱动

球泡灯驱动组件将球泡灯中常用的多款调光方案做了封装，使用一个抽象层管理这些方案，便于开发者集成到自己的应用程序中，目前所有 ESP32 系列芯片都已经完成支持。

18.1.1 调光方案

PWM 调光方案

PWM 调光是一种通过调节脉冲宽度来控制 LED 亮度的技术，核心在于通过改变电流脉冲的占空比（即高电平时间在整个周期时间中的比例）来调节。当占空比较高时，LED 获得的电流时间长，亮度高；反之，占空比低时，LED 获得的电流时间短，亮度低。所有 ESP 芯片均支持使用硬件 [LEDC 驱动](#) / [MCPWM](#) 驱动输出 PWM，推荐优先使用 LEDC 驱动实现，LEDC 驱动支持硬件渐变，频率占空比均可配置，分辨率最高为 20 bit。目前已支持下述 PWM 调光驱动类型：

- RGB + C/W
- RGB + CCT/Brightness

PWM 方案使用案例 使用 PWM 方案点亮一个 3 路 RGB 灯珠球泡灯，PWM 频率为 4000Hz，使用软件混色，启用渐变功能，上电点亮为红色。

```
lightbulb_config_t config = {  
    //1. 选择 PWM 输出并进行参数配置  
    .type = DRIVER_ESP_PWM,  
    .driver_conf.pwm.freq_hz = 4000,  
  
    //2. 功能选择，根据你的需要启用/禁用  
    .capability.enable_fade = true,  
    .capability.fade_time_ms = 800,  
    .capability.enable_lowpower = false,  
    /* 如果你的驱动白光输出为硬件单独控制而不是软件混色，需要启用此功能 */  
    .capability.enable_hardware_cct = true,  
    .capability.enable_status_storage = true,  
    /* 用于配置 LED 灯珠组合 */  
    .capability.led_beads = LED_BEADS_3CH_RGB,  
    .capability.storage_cb = NULL,  
};
```

(下页继续)

(续上页)

```

        .capability.sync_change_brightness_value = true,

//3. 配置 PWM 输出的硬件管脚
.io_conf.pwm_io.red = 25,
.io_conf.pwm_io.green = 26,
.io_conf.pwm_io.blue = 27,

//4. 限制参数
.external_limit = NULL,

//5. 颜色校准参数
.gamma_conf = NULL,

//6. 初始化照明参数, 如果 on 置位将在初始化驱动时点亮球泡灯
.init_status.mode = WORK_COLOR,
.init_status.on = true,
.init_status.hue = 0,
.init_status.saturation = 100,
.init_status.value = 100,
};
lightbulb_init(&config);

```

I2C 调光方案

I2C 调光 通过 I2C 总线发送控制命令到调光芯片, 通过改变调光芯片的输出电流, 调节 LED 的亮度。I2C 总线由两条线构成: 数据线 (SDA) 和时钟线 (SCL)。所有 ESP 芯片均支持使用硬件 I2C 通信调光芯片。目前已支持下述调光芯片:

- SM2135EH
- SM2X35EGH (SM2235EGH/SM2335EGH)
- BP57x8D (BP5758/BP5758D/BP5768)
- BP1658CJ
- KP18058

I2C 调光使用案例 使用 I2C 驱动 BP5758D 芯片点亮一个 5 路 RGBCW 灯珠球泡灯, I2C 通信速率为 300KHz, RGB 灯珠驱动电流为 50 mA, CW 灯珠驱动电流为 30mA, 使用软件混色, 启用渐变功能, 上电点亮为红色。

```

lightbulb_config_t config = {
    //1.
    ↪选择需要的芯片并进行参数配置, 每款芯片配置的参数存在不同, 请仔细参阅芯片手册
    .type = DRIVER_BP57x8D,
    .driver_conf.bp57x8d.freq_khz = 300,
    .driver_conf.bp57x8d.enable_iic_queue = true,
    .driver_conf.bp57x8d.iic_clk = 4,
    .driver_conf.bp57x8d.iic_sda = 5,
    .driver_conf.bp57x8d.current = {50, 50, 50, 30, 30},

    //2. 驱动功能选择, 根据你的需要启用/禁用
    .capability.enable_fade = true,
    .capability.fade_time_ms = 800,
    .capability.enable_lowpower = false,

    .capability.enable_status_storage = true,
    .capability.led_beads = LED_BEADS_5CH_RGBCW,
    .capability.storage_cb = NULL,
    .capability.sync_change_brightness_value = true,

```

(下页继续)

(续上页)

```

//3. 配置 IIC 芯片的硬件管脚
.io_conf.iic_io.red = OUT3,
.io_conf.iic_io.green = OUT2,
.io_conf.iic_io.blue = OUT1,
.io_conf.iic_io.cold_white = OUT5,
.io_conf.iic_io.warm_yellow = OUT4,

//4. 限制参数, 使用细则请参考后面小节
.external_limit = NULL,

//5. 颜色校准参数
.gamma_conf = NULL,

//6. 初始化照明参数, 如果 on 置位将在初始化驱动时点亮球泡灯
.init_status.mode = WORK_COLOR,
.init_status.on = true,
.init_status.hue = 0,
.init_status.saturation = 100,
.init_status.value = 100,
};
lightbulb_init(&config);

```

单线调光方案

单线调光方案是一种通过单根通信线控制 LED 亮度的方法, 核心是通过特定的通信协议发送控制信号来调节 LED 的亮度, 此类方案在 ESP 芯片上可使用 [RMT 外设](#)或 [SPI 外设](#) 实现, 推荐优先使用 [SPI](#) 实现灯珠通信控制。目前已支持下述灯珠类型:

- WS2812

WS2812 使用案例 使用 SPI 驱动点亮 10 颗 WS2812 灯珠, 启用渐变功能, 上电为红色。

```

lightbulb_config_t config = {
//1. 选择 WS2812 输出并进行参数配置
.type = DRIVER_WS2812,
.driver_conf.ws2812.led_num = 10,
.driver_conf.ws2812.ctrl_io = 4,

//2. 驱动功能选择, 根据你的需要启用/禁用
.capability.enable_fade = true,
.capability.fade_time_ms = 800,
.capability.enable_status_storage = true,

/* 对于 WS2812 只能选择 LED_BEADS_3CH_RGB */
.capability.led_beads = LED_BEADS_3CH_RGB,
.capability.storage_cb = NULL,

//3. 限制参数, 使用细则请参考后面小节
.external_limit = NULL,

//4. 颜色校准参数
.gamma_conf = NULL,

//5. 初始化照明参数, 如果 on 置位将在初始化驱动时点亮球泡灯
.init_status.mode = WORK_COLOR,
.init_status.on = true,
.init_status.hue = 0,
.init_status.saturation = 100,
.init_status.value = 100,
};

```

(下页继续)

(续上页)

```
};
lightbulb_init(&config);
```

18.1.2 渐变原理

球泡灯组件中的渐变是用软件实现的，每个通道将记录驱动输出的当前值，最终值，步长，步数等。当使用 API 设置颜色时，会更改最终值，步长，步数，并启用渐变定时器，定时器每间隔 12ms 触发一次回调，回调函数中会检查每个通道的步数，只要存在未执行完的步数，就会根据步长加减当前值，并更新到底层驱动中。当所有通道步数为 0 时，便代表渐变完成，此时将停止定时器。

18.1.3 低功耗实现流程

球泡灯如果要通过 T20 等功耗认证，优化灯板电源后，软件上需要做一些低功耗配置，除了应用 [低功耗模式使用指南](#) 中提到的配置外，驱动部分也需要进行一些处理，球泡灯组件中已经在 PWM 方案与 I2C 2 种调光驱动方案增加了相关内容，具体逻辑为开关灯时，I2C 方案调用调光芯片自身的低功耗指令，退出或进入低功耗，PWM 方案中，ESP32 由于使用 APB 时钟源，需要通过电源锁管理，否则会出现灯光闪烁，开灯时拿电源锁，禁用动态调频，关灯时释放，其他芯片采用 XTAL 时钟源，无需采取任何措施。

18.1.4 颜色校准方案

色温模式

色温模式的校准需要配置下述结构体。

```
union {
    struct {
        uint16_t kelvin_min;
        uint16_t kelvin_max;
    } standard;
    struct {
        lightbulb_cct_mapping_data_t *table;
        int table_size;
    } precise;
} cct_mix_mode;
```

- 标准模式：标定最大最小值开尔文值，使用线性插值的方式补全中间值，然后根据比色温值调节冷、暖灯珠输出的比例。
- 精准模式：校准不同色温所需要的红、绿、蓝、冷、暖灯珠输出比例，使用这些校准点直接输出对应比例，校准的点越多，色温越精准。

彩色模式

色彩模式的校准需要配置下述结构体。

```
union {
    struct {
        lightbulb_color_mapping_data_t *table;
        int table_size;
    } precise;
} color_mix_mode;
```

- 标注模式：无需参数配置，内部将使用理论算法将 HSV、XYZ 等颜色模型转化为 RGB 比例，直接按此比例点亮灯珠。
- 精准模式：使用 HSV 模型标定颜色，测量一些不同色相与饱和度颜色所需要的红、绿、蓝、冷、暖灯珠输出比例作为校准点，使用线性插值的方式补全中间值，使用这些校准点后配比点亮灯珠。

18.1.5 功率限制方案

功率限制用于平衡、微调某个通道或整体的输出电流，以满足功率需求。

如果需要对整体功率进行限制需要配置下述结构体。

```
typedef struct {
    /* Scale the incoming value
     * range: 10% <= value <= 100%
     * step: 1%
     * default min: 10%
     * default max: 100%
     */
    uint8_t white_max_brightness; /**< Maximum brightness limit for white light
    ↪output. */
    uint8_t white_min_brightness; /**< Minimum brightness limit for white light
    ↪output. */
    uint8_t color_max_value;      /**< Maximum value limit for color light output.
    ↪*/
    uint8_t color_min_value;      /**< Minimum value limit for color light output.
    ↪*/

    /* Dynamically adjust the final power
     * range: 100% <= value <= 500%
     * step: 10%
     */
    uint16_t white_max_power;      /**< Maximum power limit for white light output.
    ↪*/
    uint16_t color_max_power;      /**< Maximum power limit for color light output.
    ↪*/
} lightbulb_power_limit_t;
```

- white_max_brightness 与 white_min_brightness 用于色温模式，将色温 API 设置的 brightness 参数限制在该最大最小值之间。
- color_max_value 与 color_min_value 用于彩色模式，将彩色 API 设置的 value 参数限制在该最大最小值之间。
- white_max_power 用于色温模式限制功率，默认值为 100，即输出的最大功率为满功率的 1/2，如果为 200 则能达到冷、暖灯珠的最大功率。
- color_max_power 用于彩色模式限制功率，默认值为 100，即输出的最大功率为满功率的 1/3，如果为 300 则能达到红、绿、蓝灯珠的最大功率。

如果需要对单灯珠功率进行限制需要配置下述结构体：

```
typedef struct {
    float balance_coefficient[5]; /**< Array of float coefficients for adjusting
    ↪the intensity of each color channel (R, G, B, C, W).
                                     These coefficients help in achieving the
    ↪desired color balance for the light output. */

    float curve_coefficient;      /**< Coefficient for gamma correction. This
    ↪value is used to modify the luminance levels
                                     to suit the non-linear characteristics of
    ↪human vision, thus improving the overall
                                     visual appearance of the light. */
} lightbulb_gamma_config_t;
```

- balance_coefficient 用于微调每个灯珠输出电流，驱动最终输出将按该比例进行衰减，默认值为 1.0，即不衰减。
- curve_coefficient 用于将渐变时的线性变化转化为曲线变化。

备注：修改 balance_coefficient 将影响颜色校准的准确性，需要先调整该参数在进行颜色校准。该参数最大的用途在于，某些 I2C 调光芯片输出的电流为 5mA 或者 10mA 的整倍数，如果需要特定电流

便可使用该参数调整。

18.1.6 API 参考

Header File

- [components/led/lightbulb_driver/include/lightbulb.h](#)

Functions

`esp_err_t lightbulb_init (lightbulb_config_t *config)`

Initialize the lightbulb.

参数 **config** –Pointer to the configuration parameters for the lightbulb.

返回 `esp_err_t`

`esp_err_t lightbulb_deinit (void)`

Deinitialize the lightbulb and release resources.

返回 `esp_err_t`

`esp_err_t lightbulb_set_fade_time (uint32_t fade_time_ms)`

Set lightbulb fade time.

参数 **fade_time_ms** –Fade time in milliseconds (ms). Range: 100ms - 3000ms.

返回 `esp_err_t`

`esp_err_t lightbulb_set_fades_function (bool is_enable)`

Enable/Disable the lightbulb fade function.

参数 **is_enable** –A boolean flag indicating whether to enable (true) or disable (false) the fade function.

返回 `esp_err_t`

`esp_err_t lightbulb_set_storage_function (bool is_enable)`

Enable/Disable the lightbulb storage function.

参数 **is_enable** –A boolean flag indicating whether to enable (true) or disable (false) the storage function.

返回 `esp_err_t`

`esp_err_t lightbulb_update_status (lightbulb_status_t *new_status, bool trigger)`

Re-update the lightbulb status.

参数

- **new_status** –Pointer to the new status to be applied to the lightbulb.

- **trigger** –A boolean flag indicating whether the update should be triggered immediately.

返回 `esp_err_t`

`bool lightbulb_get_fades_function_status (void)`

Get lightbulb fade function enabled status.

返回 true if the fade function is enabled.

返回 false if the fade function is disabled.

`esp_err_t lightbulb_hsv2rgb (uint16_t hue, uint8_t saturation, uint8_t value, uint8_t *red, uint8_t *green, uint8_t *blue)`

Convert HSV model to RGB model.

备注: RGB model color depth is 8 bit (0-255).

参数

- **hue** –Hue value in the range of 0-360 degrees.
- **saturation** –Saturation value in the range of 0-100.
- **value** –Value (brightness) value in the range of 0-100.
- **red** –Pointer to a variable to store the resulting Red component (0-255).
- **green** –Pointer to a variable to store the resulting Green component (0-255).
- **blue** –Pointer to a variable to store the resulting Blue component (0-255).

返回 esp_err_t

esp_err_t **lightbulb_rgb2hsv** (uint16_t red, uint16_t green, uint16_t blue, uint16_t *hue, uint8_t *saturation, uint8_t *value)

Convert RGB model to HSV model.

备注: RGB model color depth is 8 bit (0-255).

参数

- **red** –Red component value in the range of 0-255.
- **green** –Green component value in the range of 0-255.
- **blue** –Blue component value in the range of 0-255.
- **hue** –Pointer to a variable to store the resulting Hue value (0-360 degrees).
- **saturation** –Pointer to a variable to store the resulting Saturation value (0-100).
- **value** –Pointer to a variable to store the resulting Value (brightness) value (0-100).

返回 esp_err_t

esp_err_t **lightbulb_xyy2rgb** (float x, float y, float Y, uint8_t *red, uint8_t *green, uint8_t *blue)

Convert xyY model to RGB model.

备注: Refer: <https://www.easyrgb.com/en/convert.php#inputFORM> <https://www.easyrgb.com/en/math.php>

参数

- **x** –x coordinate value in the range of 0 to 1.0.
- **y** –y coordinate value in the range of 0 to 1.0.
- **Y** –Y (luminance) value in the range of 0 to 100.0.
- **red** –Pointer to a variable to store the resulting Red component (0-255).
- **green** –Pointer to a variable to store the resulting Green component (0-255).
- **blue** –Pointer to a variable to store the resulting Blue component (0-255).

返回 esp_err_t An error code indicating the success or failure of the operation.

esp_err_t **lightbulb_rgb2xyy** (uint8_t red, uint8_t green, uint8_t blue, float *x, float *y, float *Y)

Convert RGB model to xyY model.

参数

- **red** –Red component value in the range of 0 to 255.
- **green** –Green component value in the range of 0 to 255.
- **blue** –Blue component value in the range of 0 to 255.
- **x** –Pointer to a variable to store the resulting x coordinate (0-1.0).
- **y** –Pointer to a variable to store the resulting y coordinate (0-1.0).
- **Y** –Pointer to a variable to store the resulting Y (luminance) value (0-100.0).

返回 esp_err_t An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_kelvin2percentage (uint16_t kelvin, uint8_t *percentage)`

Convert CCT (Color Temperature) kelvin to percentage.

参数

- **kelvin** –Default range: 2200k - 7000k (Color Temperature in kelvin).
- **percentage** –Pointer to a variable to store the resulting percentage (0 - 100).

返回 `esp_err_t`

`esp_err_t lightbulb_percentage2kelvin (uint8_t percentage, uint16_t *kelvin)`

Convert percentage to CCT (Color Temperature) kelvin.

Attention

参数

- **percentage** –Percentage value in the range of 0 to 100.
- **kelvin** –Pointer to a variable to store the resulting Color Temperature in kelvin. Default range: 2200k - 7000k.

返回 `esp_err_t`

`esp_err_t lightbulb_set_hue (uint16_t hue)`

Set the hue value.

参数 **hue** –Hue value in the range of 0-360 degrees.

返回 `esp_err_t` An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_set_saturation (uint8_t saturation)`

Set the saturation value.

参数 **saturation** –Saturation value in the range of 0-100.

返回 `esp_err_t` An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_set_value (uint8_t value)`

Set the value (brightness) of the lightbulb.

参数 **value** –Brightness value in the range of 0-100.

返回 `esp_err_t` An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_set_cct (uint16_t cct)`

Set the color temperature (CCT) of the lightbulb.

备注: Supports using either percentage or Kelvin values.

参数 **cct** –CCT value in the range of 0-100 or 2200-7000.

返回 `esp_err_t` An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_set_brightness (uint8_t brightness)`

Set the brightness of the lightbulb.

参数 **brightness** –Brightness value in the range of 0-100.

返回 `esp_err_t` An error code indicating the success or failure of the operation.

`esp_err_t lightbulb_set_xyY (float x, float y, float Y)`

Set the xyY color model for the lightbulb.

Attention The xyY color model cannot fully correspond to the HSV color model, so the color may be biased. The grayscale will be recalculated in lightbulb, so we cannot directly operate the underlying driver through the xyY interface.

参数

- **x** –x-coordinate value in the range of 0 to 1.0.
- **y** –y-coordinate value in the range of 0 to 1.0.
- **Y** –Y-coordinate (luminance) value in the range of 0 to 100.0.

返回 esp_err_t An error code indicating the success or failure of the operation.

esp_err_t **lightbulb_set_hsv** (uint16_t hue, uint8_t saturation, uint8_t value)

Set the HSV (Hue, Saturation, Value) color model for the lightbulb.

参数

- **hue** –Hue value in the range of 0 to 360 degrees.
- **saturation** –Saturation value in the range of 0 to 100.
- **value** –Value (brightness) value in the range of 0 to 100.

返回 esp_err_t An error code indicating the success or failure of the operation.

esp_err_t **lightbulb_set_cctb** (uint16_t cct, uint8_t brightness)

Set the color temperature (CCT) and brightness of the lightbulb.

备注: Supports using either percentage or Kelvin values.

参数

- **cct** –CCT value in the range of 0-100 or 2200-7000.
- **brightness** –Brightness value in the range of 0-100.

返回 esp_err_t An error code indicating the success or failure of the operation.

esp_err_t **lightbulb_set_switch** (bool status)

Set the on/off status of the lightbulb.

参数 **status** –On/off status (true for on, false for off).

返回 esp_err_t An error code indicating the success or failure of the operation.

int16_t **lightbulb_get_hue** (void)

Get the hue value of the lightbulb.

返回 int16_t The hue value in the range of 0 to 360 degrees.

int8_t **lightbulb_get_saturation** (void)

Get the saturation value of the lightbulb.

返回 int8_t The saturation value in the range of 0 to 100.

int8_t **lightbulb_get_value** (void)

Get the value (brightness) of the lightbulb.

返回 int8_t The brightness value in the range of 0 to 100.

int8_t **lightbulb_get_cct_percentage** (void)

Get the color temperature (CCT) percentage of the lightbulb.

返回 int8_t The CCT percentage value in the range of 0 to 100.

int16_t **lightbulb_get_cct_kelvin** (void)

Get the color temperature (CCT) Kelvin value of the lightbulb.

返回 int16_t The CCT Kelvin value in the range of 2200 to 7000.

int8_t **lightbulb_get_brightness** (void)

Get the brightness value of the lightbulb.

返回 int8_t The brightness value in the range of 0 to 100.

bool lightbulb_get_switch (void)

Get the on/off status of the lightbulb.

返回 true The lightbulb is on.

返回 false The lightbulb is off.

lightbulb_works_mode_t lightbulb_get_mode (void)

Get the work mode of the lightbulb.

返回 lightbulb_works_mode_t The current work mode of the lightbulb.

esp_err_t lightbulb_get_power_limit (*lightbulb_power_limit_t* *power_limit)

Get the power limit of the lightbulb.

参数 **power_limit** –A pointer to a *lightbulb_power_limit_t* structure where the power limit details will be stored.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_get_all_detail (*lightbulb_status_t* *status)

Get all the status details of the lightbulb.

参数 **status** –A pointer to a *lightbulb_status_t* structure where the status details will be stored.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_status_get_from_nvs (*lightbulb_status_t* *value)

Get the lightbulb status from NVS.

参数 **value** –Pointer to a *lightbulb_status_t* structure where the stored state will be read into.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_status_set_to_nvs (const *lightbulb_status_t* *value)

Store the lightbulb state to NVS.

参数 **value** –Pointer to a *lightbulb_status_t* structure representing the current running state to be stored.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_status_erase_nvs_storage (void)

Erase the lightbulb state stored in NVS.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_basic_effect_start (*lightbulb_effect_config_t* *config)

Start some blinking/breathing effects.

参数 **config** –Pointer to a *lightbulb_effect_config_t* structure containing the configuration for the effect.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_basic_effect_stop (void)

Stop the effect in progress and keep the current lighting output.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

esp_err_t lightbulb_basic_effect_stop_and_restore (void)

Stop the effect in progress and restore the previous lighting output.

返回 *esp_err_t* An error code indicating the success or failure of the operation.

void lightbulb_lighting_output_test (*lightbulb_lighting_unit_t* mask, uint16_t speed_ms)

Used to test lightbulb hardware functionality.

参数

- **mask** –A bitmask representing the test unit or combination of test units to be tested.
- **speed_ms** –The switching speed in milliseconds for the lighting patterns.

Structures

struct **lightbulb_cct_mapping_data_t**

CCT (Correlated Color Temperature) mapping data.

备注: This structure is used for precise color temperature calibration. Each color temperature value (cct_kelvin) needs to have a corresponding percentage (cct_percentage) determined, which is used to calibrate the color temperature more accurately. The rgbcw array specifies the current coefficients for the RGBCW channels. These coefficients are instrumental in adjusting the intensity of each color channel (Red, Green, Blue, Cool White, Warm White) to achieve the desired color temperature. They are also used for power limiting to ensure energy efficiency and LED longevity. Therefore, the sum of all values in the rgbcw array must equal 1 to maintain the correct power balance.

Public Members

float **rgbcw**[5]

Array of float coefficients for CCT data (R, G, B, C, W). These coefficients are used to adjust the intensity of each color channel.

uint8_t **cct_percentage**

Percentage representation of the color temperature. Used to calibrate the light's color temperature within a predefined range.

uint16_t **cct_kelvin**

The specific color temperature value in Kelvin. Used to define the perceived warmth or coolness of the light emitted.

struct **lightbulb_color_mapping_data_t**

Color mode mapping data.

备注: Used for calibrating color accuracy in color mode.

Public Members

float **rgbcw_100**[5]

The RGBCW components required when saturation is 100 at a specific hue.

float **rgbcw_50**[5]

The RGBCW components required when saturation is 50 at a specific hue.

float **rgbcw_0**[5]

The RGBCW components required when saturation is 10 at a specific hue.

uint16_t **hue**

hue.

struct **lightbulb_gamma_config_t**

Gamma correction and color balance configuration.

备注: This structure is used for calibrating the brightness proportions and color balance of a lightbulb. Typically, the human eye perceives changes in brightness in a non-linear manner, which is why gamma correction is necessary. This helps to adjust the brightness to match the non-linear perception of the human eye, ensuring a more natural and visually comfortable light output. The color balance coefficients are used to adjust the intensity of each color channel (Red, Green, Blue, Cool White, Warm White) to achieve the desired color balance and overall light quality.

Public Members

float **balance_coefficient**[5]

Array of float coefficients for adjusting the intensity of each color channel (R, G, B, C, W). These coefficients help in achieving the desired color balance for the light output.

float **color_curve_coefficient**

Coefficient for gamma correction (RGB mode). The default value is 1.0, which is linear.

float **white_curve_coefficient**

Coefficient for gamma correction (CCT mode). The default value is 1.0, which is linear.

struct **lightbulb_status_t**

The working status of the lightbulb.

Attention Both the variable `value` and the variable `brightness` are used to mark light brightness. They respectively indicate the brightness of color light and white light.

备注: Due to the differences in led beads, the percentage does not represent color temperature. The real meaning is the output ratio of cold and warm led beads: 0% lights up only the warm beads, 50% lights up an equal number of cold and warm beads, and 100% lights up only the cold beads. For simplicity, we can roughly assume that 0% represents the lowest color temperature of the beads, and 100% represents the highest color temperature. The meaning of intermediate percentages varies under different color temperature calibration schemes:

- Standard Mode: The percentage is proportionally mapped between the lowest and highest color temperatures.
 - Precise Mode:
 - For hardware CCT schemes (only applicable to PWM-driven): The percentage actually represents the duty cycle of the PWM-driven color temperature channel, with each percentage corresponding to an accurate and real color temperature.
 - For those with CW channels: The percentage represents the proportion of cold and warm beads involved in the output. The increase in percentage does not linearly correspond to the increase in color temperature, and instruments are needed for accurate determination.
 - For those using RGB channels to mix cold and warm colors: The percentage has no real significance and can be ignored. In actual use, it can serve as an index to reference corresponding color temperature values.”
-

Public Members

***lightbulb_works_mode_t* mode**

The working mode of the lightbulb (color, white, etc.).

bool on

On/off status of the lightbulb.

uint16_t hue

Hue value for color light (range: 0-360).

uint8_t saturation

Saturation value for color light (range: 0-100).

uint8_t value

Brightness value for color light (range: 0-100).

uint8_t cct_percentage

0% -> .. -> 100% 2200K -> .. -> 7000K warm -> .. -> cold Cold and warm led bead output ratio (range: 0-100).

uint8_t brightness

Brightness value for white light (range: 0-100).

struct lightbulb_power_limit_t

Output limit or gain without changing color.

Public Members**uint8_t white_max_brightness**

Maximum brightness limit for white light output.

uint8_t white_min_brightness

Minimum brightness limit for white light output.

uint8_t color_max_value

Maximum value limit for color light output.

uint8_t color_min_value

Minimum value limit for color light output.

uint16_t white_max_power

Maximum power limit for white light output.

uint16_t color_max_power

Maximum power limit for color light output.

struct lightbulb_cct_kelvin_range_t

Used to map percentages to Kelvin values.

Public Members**uint16_t min**

Minimum color temperature value in Kelvin, default is 2200 K.

uint16_t max

Maximum color temperature value in Kelvin, default is 7000 K.

struct lightbulb_capability_t

Some Lightbulb Capability Configuration Options.

Public Members**uint16_t fade_time_ms**

Fade time in milliseconds (ms), data range: 100ms - 3000ms, default is 800ms.

uint16_t storage_delay_ms

Storage delay time in milliseconds (ms), used to mitigate adverse effects of short-term repeated erasing and writing of NVS.

lightbulb_led_beads_comb_t **led_beads**

Configuration for the combination of LED beads. Please select the appropriate type for the onboard LED.

lightbulb_status_storage_cb_t **storage_cb**

Callback function to be called when the lightbulb status starts to be stored.

bool enable_fade

Enable this option to use fade effects for color switching instead of direct rapid changes.

bool enable_lowpower

Enable low-power regulation when the lights are off.

bool enable_status_storage

Enable this option to store the lightbulb state in NVS.

bool enable_hardware_cct

Enable this option if your driver uses hardware CCT. Some PWM type drivers may need to set this option.

bool enable_precise_cct_control

Enable this option if you need precise CCT control. Must set 'enable_hardware_cct' to false in order to enable it.

bool enable_precise_color_control

Enable this option if you need precise Color control.

bool sync_change_brightness_value

Enable this option if you need to use a parameter to mark the brightness of the white and color output.

bool **disable_auto_on**

Enable this option if you don't need automatic on when the color/white value is set.

struct **lightbulb_config_t**

Lightbulb Configuration Options.

Public Members

lightbulb_driver_t **type**

Type of the lightbulb driver.

union *lightbulb_config_t*::[anonymous] **driver_conf**

Configuration specific to the lightbulb driver.

uint16_t **kelvin_min**

Minimum Kelvin value.

uint16_t **kelvin_max**

Maximum Kelvin value.

struct *lightbulb_config_t*::[anonymous]::[anonymous] **standard**

Standard Mode

lightbulb_cct_mapping_data_t ***table**

Mixed Color table

int **table_size**

Table size

struct *lightbulb_config_t*::[anonymous]::[anonymous] **precise**

Precise Mode

union *lightbulb_config_t*::[anonymous] **cct_mix_mode**

This configuration is used to set up the CCT (Correlated Color Temperature) calibration scheme. The default mode is the standard mode, which requires no additional configuration. For the precise mode, a color mixing table needs to be configured. To enable this precise CCT control, the 'enable_precise_cct_control' should be set to true in the capability settings.

lightbulb_color_mapping_data_t ***table**

Mixed Color table

struct *lightbulb_config_t*::[anonymous]::[anonymous] **precise**

Precise Mode

union *lightbulb_config_t*::[anonymous] **color_mix_mode**

This configuration is used to set up the color calibration scheme. Measure certain hue and saturation values as calibration points, and use a linear interpolation method for color calibration.

lightbulb_gamma_config_t ***gamma_conf**

Pointer to the gamma configuration data.

lightbulb_power_limit_t ***external_limit**

Pointer to the external power limit configuration.

gpio_num_t **red**

GPIO Pin for the red LED

gpio_num_t **green**

GPIO Pin for the green LED

gpio_num_t **blue**

GPIO Pin for the blue LED

gpio_num_t **cold_cct**

GPIO Pin for the cold or cct LED

gpio_num_t **warm_brightness**

GPIO Pin for the warm or brightness LED

struct *lightbulb_config_t*::[anonymous]::[anonymous] **pwm_io**

Configuration for PWM driver I/O pins.

lightbulb_chip_out_pin_t **red**

IIC dimming driver output channel for red LED

GPIO Pin for the red LED for SM16825E driver

lightbulb_chip_out_pin_t **green**

IIC dimming driver output channel for green LED

GPIO Pin for the green LED for SM16825E driver

lightbulb_chip_out_pin_t **blue**

IIC dimming driver output channel for blue LED

GPIO Pin for the blue LED for SM16825E driver

lightbulb_chip_out_pin_t **cold_white**

IIC dimming driver output channel for cold or white LED

GPIO Pin for the white LED for SM16825E driver

lightbulb_chip_out_pin_t **warm_yellow**

IIC dimming driver output channel for warm or yellow LED

GPIO Pin for the yellow LED for SM16825E driver

struct *lightbulb_config_t*::[anonymous]::[anonymous] **iic_io**

Configuration for IIC driver I/O pins.

struct *lightbulb_config_t*::[anonymous]::[anonymous] **sm16825e_io**
Configuration for SM16825E driver I/O pins.

union *lightbulb_config_t*::[anonymous] **io_conf**
Union for I/O configuration based on the selected driver type.

lightbulb_capability_t **capability**
Lightbulb capability configuration.

lightbulb_status_t **init_status**
Initial status of the lightbulb.

struct **lightbulb_effect_config_t**
Effect function configuration options.

Public Members

lightbulb_effect_t **effect_type**
Type of the effect to be configured.

lightbulb_works_mode_t **mode**
Working mode of the lightbulb during the effect.

uint16_t **hue**
Hue component value for the effect (0-360).

uint8_t **saturation**
Saturation component value for the effect (0-100).

uint16_t **cct**
Color temperature value for the effect.

uint8_t **min_value_brightness**
Minimum brightness level for the effect (0-100).

uint8_t **max_value_brightness**
Maximum brightness level for the effect (0-100).

uint16_t **effect_cycle_ms**
Cycle time for the effect in milliseconds (ms).

int **total_ms**
Total duration of the effect in milliseconds (ms). If greater than 0, enables an auto-stop timer.

void (***user_cb**)(void)
User-defined callback function to be called when the auto-stop timer triggers.

bool **interrupt_forbidden**
If true, the auto-stop timer can only be stopped by specific interfaces or FreeRTOS triggers.

Type Definitions

typedef esp_err_t (***lightbulb_status_storage_cb_t**)(*lightbulb_status_t* status)

Function pointer type to store lightbulb status.

Param status The *lightbulb_status_t* structure containing the current status of the lightbulb.

Return esp_err_t

typedef *lightbulb_chip_out_pin_t* **lightbulb_iic_out_pin_t**

IIC dimming chip output port definition.

Enumerations

enum **lightbulb_driver_t**

Supported drivers.

Values:

enumerator **DRIVER_SELECT_INVALID**

enumerator **DRIVER_ESP_PWM**

enumerator **DRIVER_SM2135E**

enumerator **DRIVER_SM2135EH**

enumerator **DRIVER_SM2182E**

enumerator **DRIVER_SM2x35EGH**

enumerator **DRIVER_BP1658CJ**

enumerator **DRIVER_BP57x8D**

enumerator **DRIVER_KP18058**

enumerator **DRIVER_WS2812**

enumerator **DRIVER_SM16825E**

enumerator **DRIVER_SELECT_MAX**

enum **lightbulb_led_beads_comb_t**

Supported LED bead combinations.

Values:

enumerator **LED_BEADS_INVALID**

Invalid LED bead combination.

enumerator **LED_BEADS_1CH_C**

Single-channel: Cold white LED bead.

enumerator **LED_BEADS_1CH_W**

Single-channel: Warm white LED bead.

enumerator **LED_BEADS_2CH_CW**

Two channels: Warm white + cold white LED beads combination.

enumerator **LED_BEADS_3CH_RGB**

Three channels: Red + green + blue LED beads combination.

enumerator **LED_BEADS_4CH_RGBC**

Four channels: Red + green + blue + cold white LED beads combination.

enumerator **LED_BEADS_4CH_RGBCC**

Four channels: Red + green + blue + 2 * cold white LED beads combination.

enumerator **LED_BEADS_4CH_RGBW**

Four channels: Red + green + blue + warm white LED beads combination.

enumerator **LED_BEADS_4CH_RGBWW**

Four channels: Red + green + blue + 2 * warm white LED beads combination.

enumerator **LED_BEADS_5CH_RGBCW**

Five channels: Red + green + blue + cold white + warm white LED beads combination.

enumerator **LED_BEADS_5CH_RGBCC**

Five channels: Red + green + blue + 2 * cold white + RGB mix warm white LED beads combination.

enumerator **LED_BEADS_5CH_RGBWW**

Five channels: Red + green + blue + 2 * warm white + RGB mix cold white LED beads combination.

enumerator **LED_BEADS_5CH_RGBC**

Five channels: Red + green + blue + cold white + RGB mix warm white LED beads combination.

enumerator **LED_BEADS_5CH_RGBW**

Five channels: Red + green + blue + warm white + RGB mix cold white LED beads combination.

enumerator **LED_BEADS_MAX**

Maximum number of supported LED bead combinations.

enum **lightbulb_effect_t**

Supported effects.

Values:

enumerator **EFFECT_BREATH**

enumerator **EFFECT_BLINK**

enum **lightbulb_lighting_unit_t**

Lighting test units.

Values:

enumerator **LIGHTING_RAINBOW**

Rainbow lighting effect.

enumerator **LIGHTING_WARM_TO_COLD**

Transition from warm to cold lighting.

enumerator **LIGHTING_COLD_TO_WARM**

Transition from cold to warm lighting.

enumerator **LIGHTING_BASIC_FIVE**

Basic five lighting colors.

enumerator **LIGHTING_COLOR_MUTUAL_WHITE**

Color and mutual white lighting.

enumerator **LIGHTING_COLOR_EFFECT**

Color-specific lighting effect.

enumerator **LIGHTING_WHITE_EFFECT**

White-specific lighting effect.

enumerator **LIGHTING_ALEXA**

Alexa integration lighting.

enumerator **LIGHTING_COLOR_VALUE_INCREMENT**

Incrementing color values lighting.

enumerator **LIGHTING_WHITE_BRIGHTNESS_INCREMENT**

Incrementing white brightness lighting.

enumerator **LIGHTING_ALL_UNIT**

All lighting units.

enum **lightbulb_works_mode_t**

The working mode of the lightbulb.

Values:

enumerator **WORK_INVALID**

Invalid working mode.

enumerator **WORK_COLOR**

Color mode, where the lightbulb emits colored light.

enumerator **WORK_WHITE**

White mode, where the lightbulb emits white light.

enum **lightbulb_chip_out_pin_t**

Port enumeration names for multiple output channel chips.

Values:

enumerator **OUT1**

Chip output port 1/RED.

enumerator **OUT2**

Chip output port 2/GREEN.

enumerator **OUT3**

Chip output port 3/BLUE.

enumerator **OUT4**

Chip output port 4/WHITE.

enumerator **OUT5**

Chip output port 5/YELLOW.

enumerator **OUT_MAX**

The maximum value for the Chip output port enumeration, this is invalid value.

Chapter 19

解决方案

19.1 4G 解决方案

19.1.1 IOT ETH 以太网

iot_eth 提供了一整套 4G 模组连接以太网的解决方案，提供底层硬件的抽象层，默认提供基于 LWIP 的网络接口。也支持用户自定义网络接口。

特性

- 高级工厂模式，支持底层多种连接方式
- 默认支持 LWIP 网络接口, 也支持用户自定义网络接口

用户指南

1. 先初始化 *driver* 层，提供下层硬件抽象

```
iot_usbh_rndis_config_t rndis_cfg = {
    .auto_detect = true,    // auto detect 4G module
    .auto_detect_timeout = pdMS_TO_TICKS(1000), // auto detect timeout
};

iot_eth_driver_t *rndis_handle = NULL;
esp_err_t ret = iot_eth_new_usb_rndis(&rndis_cfg, &rndis_handle);
if (ret != ESP_OK || rndis_handle == NULL) {
    ESP_LOGE(TAG, "Failed to create USB RNDIS driver");
    return;
}
```

2. 初始化 *iot_eth* 层

```
iot_eth_config_t eth_cfg = {
    .driver = rndis_handle,
    .stack_input = NULL,
```

(下页继续)

(续上页)

```

        .user_data = NULL,
    };

    iot_eth_handle_t eth_handle = NULL;
    ret = iot_eth_install(&eth_cfg, &eth_handle);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to install USB RNDIS driver");
        return;
    }

```

3. 启动 *iot_eth* 层

```

ret = iot_eth_start(eth_handle);
if (ret != ESP_OK) {
    ESP_LOGE(TAG, "Failed to start USB RNDIS driver");
    return;
}

```

使用其他网络接口

1. 发送：请调用 *iot_eth_transmit* 发送数据。
2. 接受：请配置 *iot_eth_config_t* 中的 *stack_input* 回调函数，当有数据接收时，会调用该回调函数。
3. 获取 4G 模块的 MAC 地址：请调用 *iot_eth_get_addr* 获取 4G 模块的 MAC 地址。

已支持的网络接口

- 基于 USB 接口的 RNDIS *USB RNDIS* 主机驱动
- 基于 USB 接口的 PPP *USB PPP*
- 基于 USB 接口的 ECM *USB ECM* 主机驱动

API 参考

Header File

- [components/iot_eth/include/iot_eth.h](#)

Functions

`esp_err_t iot_eth_install (iot_eth_config_t *config, iot_eth_handle_t *handle)`

`esp_err_t iot_eth_uninstall (iot_eth_handle_t handle)`

`esp_err_t iot_eth_start (iot_eth_handle_t handle)`

`esp_err_t iot_eth_stop (iot_eth_handle_t handle)`

`esp_err_t iot_eth_transmit (iot_eth_handle_t handle, void *data, size_t len)`

`esp_err_t iot_eth_get_addr (iot_eth_handle_t handle, uint8_t *mac)`

`esp_err_t iot_eth_update_input_path (iot_eth_handle_t handle, static_input_cb_t stack_input, void *user_data)`

Structures

`struct iot_eth_config_t`

Ethernet configuration structure.

This structure holds the configuration for initializing an Ethernet instance.

Public Members

`iot_eth_driver_t *driver`

Pointer to the Ethernet driver

`static_input_cb_t stack_input`

Function pointer for stack input

`void *user_data`

User data for callbacks

Type Definitions

`typedef void *iot_eth_handle_t`

iot_eth event base declaration

Ethernet handle type

This is a handle to an Ethernet instance, used for managing Ethernet operations.

`typedef esp_err_t (*static_input_cb_t)(iot_eth_handle_t handle, uint8_t *data, size_t len, void *user_data)`

Static input callback function type.

This type defines a function pointer for a static input callback. It takes an Ethernet handle, a pointer to data, the length of the data, and a user data pointer as arguments.

Enumerations

`enum iot_eth_event_t`

iot_eth event declarations

Values:

enumerator `IOT_ETH_EVENT_START`

IOT_ETH driver start

enumerator `IOT_ETH_EVENT_STOP`

IOT_ETH driver stop

enumerator `IOT_ETH_EVENT_CONNECTED`

IOT_ETH got a valid link

enumerator `IOT_ETH_EVENT_DISCONNECTED`

IOT_ETH lost a valid link

Chapter 20

其它资源

20.1 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。

20.1.1 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	规格书	mcp23017

20.2 ADC 扩展量程方案

重要：当前 ADC 扩展量程方案仅适用于 ESP32-S2 与 ESP32-S3 芯片。

ESP32-S2/S3 ADC 在默认量程范围可以通过外部分压电路，实现大部分的 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-S2/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-S2/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 设定电压，直接输出测量电压值作为测量结果
3. 当测量电压值大于 设定电压，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。

备注：对于 ESP32-S2 而言，设定电压为 2600 mV；对于 ESP32-S3 而言，设定电压为 2900 mV。

20.2.1 Patch 使用方法

重要：当前补丁基于 ESP-IDF v4.4.8 与 v5.3.1 开发，如需在其他 ESP-IDF 版本上使用该方案，请参考补丁内容自行修改 ESP-IDF。

基于 ESP-IDF v4.4.8 的 Patch 使用方法

- ESP32-S2 ADC 扩展补丁下载：esp32s2_adc_range_to_3300.patch
- ESP32-S3 ADC 扩展补丁下载：esp32s3_adc_range_to_3300.patch

以 ESP32-S3 为例，请按照如下方式加载补丁：

1. 确认 ESP-IDF 已经 checkout 到 v4.4.8
2. 使用指令 `git am --signoff < esp32s3_adc_range_to_3300.patch` 将 Patch 应用到 IDF 中
3. 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

基于 ESP-IDF v5.3.1 的 Patch 使用方法

- ESP32-S2 ADC 扩展补丁下载：esp32s2_adc_range_to_3300_v531.patch
- ESP32-S3 ADC 扩展补丁下载：esp32s3_adc_range_to_3300_v531.patch

以 ESP32-S3 为例，请按照如下方式加载补丁：

1. 确认 ESP-IDF 已经 checkout 到 v5.3.1
2. 使用指令 `git am --signoff < esp32s3_adc_range_to_3300_v531.patch` 将 Patch 应用到 IDF 中
3. 请注意，该方案仅对 `adc_oneshot_get_calibrated_result` 接口有效，用户可直接调用该接口获取扩展后的读数

20.2.2 API 使用说明

对于不同版本的 ESP-IDF，获取 ADC 量程扩展后的电压值方法有所不同：

- ESP-IDF v4.4.8
 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 v4.4.8 ADC 其它 API 不受影响，读取的结果和默认结果一致
- ESP-IDF v5.3.1
 1. 如果想要读取量程扩展后的电压值，用户必须使用 `adc_oneshot_get_calibrated_result` 直接获取 ADC1 或 ADC2 的通道电压
 2. ESP-IDF v5.3.1 ADC 其它 API 不受影响，读取的结果和默认结果一致

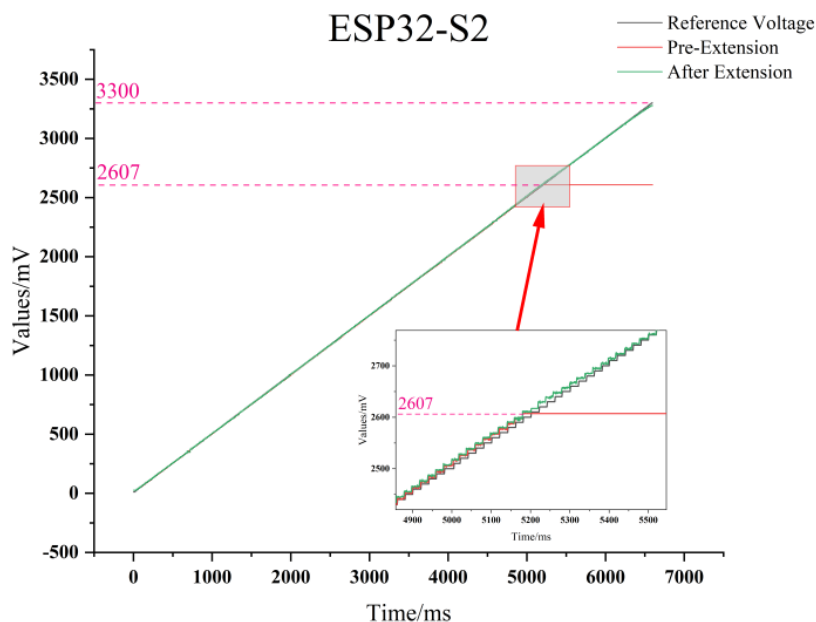
补丁默认开启 ADC 量程扩展，并默认使用 `float` 数据类型进行校正，若您需要关闭 ADC 扩展或更换校正过程中的数据类型，请参考如下过程：

- 对于 ESP-IDF v4.4.8：请使用 `menuconfig` 在 `Component config` → `Driver configurations` → `ADC configuration` → `ADC User Code Offset` 进行修改。

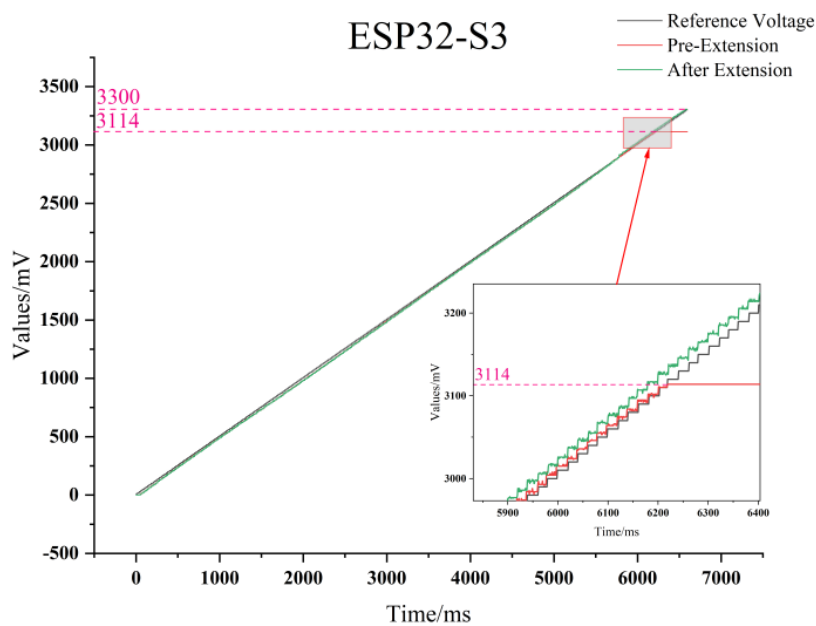
- 对于 ESP-IDF v5.3.1: 请使用 menuconfig 在 Component config → ADC and ADC Calibration → ADC User Code Offset 进行修改。

20.2.3 ADC 量程扩展效果对比

ESP32-S2 ADC 量程扩展效果对比:



ESP32-S3 ADC 量程扩展效果对比:



20.3 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。

20.3.1 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

20.3.2 配置项

- USE_GPTIMER：决定是否使用 GPTimer 驱动默认使用 ESPTimer

20.3.3 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:
    }
```

(下页继续)

```

        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);

```

20.3.4 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

void **zero_detect_pause** ([zero_detect_handle_t](#) zcd_handle)

Pause zero detect.

参数 zcd_handle –A zero detect handle

void **zero_detect_resume** ([zero_detect_handle_t](#) zcd_handle)

Resume zero detect.

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

```
zero\_signal\_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

```
zero\_signal\_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

zero_signal_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_signal_edge_t**

Zero detection signal edge type.

Values:

enumerator **CAP_EDGE_POS**

enumerator **CAP_EDGE_NEG**

enum **zero_driver_type_t**

Zero detection driver type.

Values:

enumerator **GPIO_TYPE**

Use GPIO as driver

20.4 ADC 两点校准方案

重要： 当前 ADC 两点校准方案仅适用于 ESP32 与 ESP32-S2 芯片

ESP32 出厂时默认采用参考电压进行 ADC 校准。如果用户希望进一步提高 ADC 测量结果的一致性，可以将 eFuse 中的校准方案更改为两点校准方法。

同样，对于 ESP32-S2 芯片，出厂默认使用的是两点校准法（方法一）。然而，这种方法并未考虑不同 ADC 衰减下的数据进行校准。如果用户希望使用不同衰减下的数据进行校准，可以将 eFuse 中的校准方案更改为两点校准方法（方法二）。

需要注意的是，上述两种方法都需要用户使用 `espefuse` 工具来修改 eFuse 中的校准方案，并输入相应的校准数据。这个过程需要格外谨慎，一旦烧录了错误的校准数据，将无法再次修改。这是由于 eFuse 中的数据是不可逆的，一旦写入就无法更改。因此，在进行这些操作时，请确保数据的准确性，以避免不必要的损失。

为避免上述修改 eFuse 带来的风险，`adc_1p_calibration` 组件提供了一种允许用户在应用层进行校准的方法，具备以下特点：

- 支持在应用层填入校准参数，用户可自行将校准存储到 NVS、SD 卡等存储介质中
- 支持 ESP32 使用两点校准方案，ESP32-S2 使用两点校准方案（方法二），且不影响现有 ESP-IDF 的校准方案

20.4.1 ESP32 两点校准原理

ESP32 两点校准需要分别向 ADC1 通道与 ADC2 通道施加稳定的 150 mV 与 850 mV 电压，使用 ADC 获取原始数据接口获取当前电压下的稳定 ADC 原始数据：

Input Voltage	ADC1 (ATTEN0)	ADC2 (ATTEN0)
150 mV	A1	A2
850 mV	B1	B2

备注：获取 150 mV 与 850 mV 下的 ADC 原始数据时，需要配置 ADC 驱动的衰减为 ADC_ATTEN_DB_0

ESP32 ADC 两点校准计算公式为：

$$y = \frac{coeff_a * x + coeff_{around}}{coeff_{scale}} + coeff_b$$

- $coeff_a$: $\frac{(850-150)*atten_{scale}[atten]+(B-A)/2}{B-A}$
- $coeff_b$: $850 - \frac{(850-150)*B+(B-A)/2}{B-A} + atten_{offset}[atten]$
- $coeff_{scale}$: 65536
- $coeff_{around}$: $\frac{coeff_{scale}}{2}$

ADC1	ATTEN0	ATTEN_2_5	ATTEN_6	ATTEN_12
$atten_{scale}$	65504	86975	120389	224310
$atten_{offset}$	0	1	27	54

ADC2	ATTEN0	ATTEN_2_5	ATTEN_6	ATTEN_12
$atten_{scale}$	65467	86861	120416	224708
$atten_{offset}$	0	9	26	66

20.4.2 ESP32-S2 两点校准（方法二）原理

ESP32-S2 两点校准（方法二）需要分别向 ADC1 通道与 ADC2 通道施加稳定的 600 mV、800 mV、1000 mV 与 2000 mV 电压，使用 ADC 获取原始数据接口获取当前电压下的稳定 ADC 原始数据，以 ADC1 为例：

Input Voltage	ATTEN	ADC1 Raw Value
600 mV	ATTEN_0	High[ATTEN_0]
800 mV	ATTEN_2_5	High[ATTEN_2_5]
1000 mV	ATTEN_6	High[ATTEN_6]
2000 mV	ATTEN_12	High[ATTEN_12]

备注：ESP32-S2 两点校准（方法二）的校准点选取需要分别在对应的衰减与标定电压下获取稳定的 ADC 原始数据，以确保校准的准确性。其中 High 用于存储不同衰减与标定电压下的 ADC 原始数据

ESP32-S2 ADC 两点校准（方法二）计算公式为：

$$y = \frac{x * coeff_a / (coeff_{ascaling} / coeff_{bscaling}) + coeff_b}{coeff_{bscaling}}$$

- $coeff_a$: $coeff_{ascaling} * v_{high}[atten] / High[atten]$
- $coeff_b$: 0
- $coeff_{ascaling}$: 65536
- $coeff_{bscaling}$: 1024

	ATTEN0	ATTEN_2_5	ATTEN_6	ATTEN_12
v_{high}	600	800	1000	2000

20.4.3 API 参考

Header File

- [components/utilities/adc_tp_calibration/include/adc_tp_calibration.h](#)

Functions

adc_tp_cali_handle_t **adc_tp_cali_create** (*adc_tp_cali_config_t* *config, adc_atten_t atten)

Create ADC calibration handle.

参数

- **config** –Pointer to ADC calibration configuration
- **atten** –ADC attenuation level

返回 *adc_tp_cali_handle_t* Handle of ADC calibration, NULL if failed

esp_err_t **adc_tp_cali_delete** (*adc_tp_cali_handle_t* *adc_tp_cali_handle)

Delete ADC calibration handle.

参数 **adc_tp_cali_handle** –Pointer to ADC calibration handle

返回 esp_err_t

- ESP_OK: Always returns ESP_OK

esp_err_t **adc_tp_cali_raw_to_voltage** (*adc_tp_cali_handle_t* adc_tp_cali_handle, int raw_value, int *voltage)

Convert ADC raw value to voltage in millivolts.

参数

- **adc_tp_cali_handle** –ADC calibration handle
- **raw_value** –ADC raw value to convert
- **voltage** –Pointer to store the converted voltage in millivolts

返回 esp_err_t

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid argument

Structures

struct **adc_tp_cali_config_t**

ADC calibration configuration structure.

This structure contains the raw ADC values measured at specific voltage points for calibration. The values are different for ESP32 and ESP32-S2.

Public Members

adc_unit_t **adc_unit**

ADC unit (ADC1 or ADC2)

Type Definitions

typedef void ***adc_tp_cali_handle_t**

ADC calibration handle

Chapter 21

Contributions Guide

We welcome contributions to the esp-iot-solution project!

21.1 How to Contribute

Contributions to esp-iot-solution - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via [GitHub Pull Requests](#).

21.2 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.

21.3 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.

21.4 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.

21.5 Related Documents

21.5.1 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，或者已经达成共识的符号或词语缩写；
- 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	abbreviations-in-code	<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

A

- adc_tp_cali_config_t (C++ struct), 472
- adc_tp_cali_config_t::adc_unit (C++ member), 472
- adc_tp_cali_create (C++ function), 472
- adc_tp_cali_delete (C++ function), 472
- adc_tp_cali_handle_t (C++ type), 472
- adc_tp_cali_raw_to_voltage (C++ function), 472
- ALARM_COUNT_US (C macro), 411
- ALIGNMENTDUTY (C macro), 412
- ALIGNMENTNMS (C macro), 412
- audio_codec_adc_cfg_t (C++ struct), 264
- audio_codec_adc_cfg_t::adc_channel_list (C++ member), 264
- audio_codec_adc_cfg_t::adc_channel_num (C++ member), 264
- audio_codec_adc_cfg_t::atten (C++ member), 264
- audio_codec_adc_cfg_t::conv_frame_size (C++ member), 264
- audio_codec_adc_cfg_t::handle (C++ member), 264
- audio_codec_adc_cfg_t::max_store_buf_size (C++ member), 264
- audio_codec_adc_cfg_t::sample_rate_hz (C++ member), 264
- audio_codec_adc_cfg_t::unit_id (C++ member), 264
- audio_codec_new_adc_data (C++ function), 264
- audio_frame_format (C++ enum), 269
- audio_frame_format::FORMAT_PCM (C++ enumerator), 269
- audio_frame_info_t (C++ struct), 267
- audio_frame_info_t::bits_per_sample (C++ member), 267
- audio_frame_info_t::channel (C++ member), 267
- audio_frame_info_t::format (C++ member), 267
- audio_frame_info_t::sample_rate (C++ member), 267
- audio_set_clock_cb (C++ type), 269
- audio_write_cb (C++ type), 269
- avi_play_end_cb (C++ type), 269
- avi_player_config_t (C++ struct), 268
- avi_player_config_t::audio_cb (C++ member), 268
- avi_player_config_t::audio_set_clock_cb (C++ member), 268
- avi_player_config_t::avi_play_end_cb (C++ member), 268
- avi_player_config_t::buffer_size (C++ member), 268
- avi_player_config_t::coreID (C++ member), 268
- avi_player_config_t::priority (C++ member), 268
- avi_player_config_t::stack_in_psrnm (C++ member), 268
- avi_player_config_t::stack_size (C++ member), 268
- avi_player_config_t::user_data (C++ member), 268
- avi_player_config_t::video_cb (C++ member), 268
- avi_player_deinit (C++ function), 266
- avi_player_get_audio_buffer (C++ function), 266
- avi_player_get_video_buffer (C++ function), 266
- avi_player_handle_t (C++ type), 269
- avi_player_init (C++ function), 266
- avi_player_play_from_file (C++ function), 265
- avi_player_play_from_memory (C++ function), 265
- avi_player_play_stop (C++ function), 266
- AVOID_CONTINUE_CURRENT_TIME (C macro), 413

B

- BAS_CHR_BATTERY_INFO_FEATURE_BM_NONE (C macro), 54
- BAS_CHR_BATTERY_INFO_FEATURE_BM_RECHARGE (C macro), 55
- BAS_CHR_BATTERY_INFO_FEATURE_BM_REPLACE (C macro), 55
- BAS_CHR_BATTERY_INFO_FEATURE_BM_RFU (C macro), 55
- BAS_CHR_BATTERY_INFO_FLAGS_BM_AGGREGATION_GROUP (C macro), 54

BAS_CHR_LEVEL_STATUS_POWER_STATE_CHARGE_TYPE_1 (C++ enum), 410
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_CHARGE_TYPE_UNKNOWN (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_CHARGE_TYPE_VOLTAGE (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRED_EXTERNAL_POWER_SOURCE (C++ function), 408
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRED_EXTERNAL_POWER_SOURCE_NOT_CONNECTED (C++ function), 409
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRED_EXTERNAL_POWER_SOURCE_RFU (C++ function), 408
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRED_EXTERNAL_POWER_SOURCE_RFU (C++ function), 408
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRED_EXTERNAL_POWER_SOURCE_RFU (C++ function), 408
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRELESS_EXTERNAL_POWER_SOURCE_CONNECTED (C++ member), 409
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRELESS_EXTERNAL_POWER_SOURCE_DISCONNECTED (C++ member), 409
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRELESS_EXTERNAL_POWER_SOURCE_RFU (C macro), 413
 (C macro), 52
 BAS_CHR_LEVEL_STATUS_POWER_STATE_WIRELESS_EXTERNAL_POWER_SOURCE_UNKNOWN (C macro), 82
 (C macro), 52
 BAS_CHR_TIME_STATUS_FLAGS_BM_DISCHARGED (C macro), 82
 (C macro), 53
 BAS_CHR_TIME_STATUS_FLAGS_BM_NONE (C macro), 81
 (C macro), 53
 BAS_CHR_TIME_STATUS_FLAGS_BM_RECHARGE (C macro), 82
 (C macro), 54
 BAS_CHR_TIME_STATUS_FLAGS_BM_RFU (C macro), 82
 (C macro), 54
 BASE_SPEED (C macro), 413
 BASE_VOLTAGE (C macro), 413
 bldc_control_config_t (C++ struct), 409
 bldc_control_config_t::alignment_mode (C++ member), 409
 bldc_control_config_t::control_mode (C++ member), 409
 bldc_control_config_t::debug_config (C++ member), 410
 bldc_control_config_t::six_step_config (C++ member), 409
 bldc_control_config_t::speed_mode (C++ member), 409
 bldc_control_config_t::zero_cross_comparator_config (C++ member), 410
 bldc_control_deinit (C++ function), 408
 bldc_control_event_t (C++ enum), 410
 bldc_control_event_t::BLDC_CONTROL_ALIGNMENT (C++ enumerator), 410
 bldc_control_event_t::BLDC_CONTROL_BLOCKED (C++ enumerator), 410
 bldc_control_event_t::BLDC_CONTROL_CLOSED_LOOP (C++ enumerator), 410
 bldc_control_event_t::BLDC_CONTROL_DRAG (C++ enumerator), 410
 bldc_control_event_t::BLDC_CONTROL_START (C++ enumerator), 410
 bldc_control_event_t::BLDC_CONTROL_STOP (C++ enumerator), 410
 bldc_control_get_dir (C++ function), 408
 bldc_control_get_duty (C++ function), 408
 bldc_control_get_speed_rpm (C++ function), 409
 bldc_control_handle_t (C++ type), 410
 bldc_control_set_dir (C++ function), 408
 bldc_control_set_speed_rpm (C++ function), 409
 bldc_control_start (C++ function), 408
 bldc_debug_config_t (C++ struct), 409
 BLE_ANP_CAT_BM_EMAIL (C macro), 82
 BLE_ANP_CAT_BM_MISSED_CALL (C macro), 82
 BLE_ANP_CAT_BM_NEWS (C macro), 82
 BLE_ANP_CAT_BM_NONE (C macro), 81
 BLE_ANP_CAT_BM_SCHEDULE (C macro), 82
 BLE_ANP_CAT_BM_SIMPLE_ALERT (C macro), 82
 BLE_ANP_CAT_BM_SMS (C macro), 82
 BLE_ANP_CAT_BM_VOICE_MAIL (C macro), 82
 BLE_ANP_CAT_ID_CALL (C macro), 82
 BLE_ANP_CAT_ID_EMAIL (C macro), 82
 BLE_ANP_CAT_ID_MISSED_CALL (C macro), 82
 BLE_ANP_CAT_ID_NEWS (C macro), 82
 BLE_ANP_CAT_ID_SCHEDULE (C macro), 82
 BLE_ANP_CAT_ID_SIMPLE_ALERT (C macro), 82
 BLE_ANP_CAT_ID_SMS (C macro), 82
 BLE_ANP_CAT_ID_VOICE_MAIL (C macro), 82
 BLE_ANP_CAT_NUM (C macro), 82
 BLE_ANP_CHR_UUID16_ALERT_NOT_CTRL_PT (C macro), 81
 BLE_ANP_CHR_UUID16_NEW_ALERT (C macro), 81
 BLE_ANP_CHR_UUID16_SUP_NEW_ALERT_CAT (C macro), 81
 BLE_ANP_CHR_UUID16_SUP_UNR_ALERT_CAT (C macro), 81
 BLE_ANP_CHR_UUID16_UNR_ALERT_STAT (C macro), 81
 BLE_ANP_CMD_DIS_NEW_ALERT_CAT (C macro), 82
 BLE_ANP_CMD_DIS_UNR_ALERT_CAT (C macro), 82
 BLE_ANP_CMD_EN_NEW_ALERT_CAT (C macro), 82
 BLE_ANP_CMD_EN_UNR_ALERT_CAT (C macro), 82
 BLE_ANP_CMD_NOT_NEW_ALERT_IMMEDIATE (C macro), 82

- BLE_ANP_CMD_NOT_UNR_ALERT_IMMEDIATE (C macro), 82
- BLE_ANP_INFO_STR_MAX_LEN (C macro), 83
- BLE_ANP_NEW_ALERT_MAX_LEN (C macro), 83
- BLE_ANP_UUID16 (C macro), 81
- BLE_ANS_CAT_BM_CALL (C macro), 40
- BLE_ANS_CAT_BM_EMAIL (C macro), 40
- BLE_ANS_CAT_BM_MISSED_CALL (C macro), 40
- BLE_ANS_CAT_BM_NEWS (C macro), 40
- BLE_ANS_CAT_BM_NONE (C macro), 40
- BLE_ANS_CAT_BM_SCHEDULE (C macro), 40
- BLE_ANS_CAT_BM_SIMPLE_ALERT (C macro), 40
- BLE_ANS_CAT_BM_SMS (C macro), 40
- BLE_ANS_CAT_BM_VOICE_MAIL (C macro), 40
- BLE_ANS_CAT_ID_CALL (C macro), 41
- BLE_ANS_CAT_ID_EMAIL (C macro), 40
- BLE_ANS_CAT_ID_MISSED_CALL (C macro), 41
- BLE_ANS_CAT_ID_NEWS (C macro), 41
- BLE_ANS_CAT_ID_SCHEDULE (C macro), 41
- BLE_ANS_CAT_ID_SIMPLE_ALERT (C macro), 40
- BLE_ANS_CAT_ID_SMS (C macro), 41
- BLE_ANS_CAT_ID_VOICE_MAIL (C macro), 41
- BLE_ANS_CAT_NUM (C macro), 41
- BLE_ANS_CHR_UUID16_ALERT_NOT_CTRL_PT (C macro), 40
- BLE_ANS_CHR_UUID16_NEW_ALERT (C macro), 40
- BLE_ANS_CHR_UUID16_SUP_NEW_ALERT_CAT (C macro), 40
- BLE_ANS_CHR_UUID16_SUP_UNR_ALERT_CAT (C macro), 40
- BLE_ANS_CHR_UUID16_UNR_ALERT_STAT (C macro), 40
- BLE_ANS_CMD_DIS_NEW_ALERT_CAT (C macro), 41
- BLE_ANS_CMD_DIS_UNR_ALERT_CAT (C macro), 41
- BLE_ANS_CMD_EN_NEW_ALERT_CAT (C macro), 41
- BLE_ANS_CMD_EN_UNR_ALERT_CAT (C macro), 41
- BLE_ANS_CMD_NOT_NEW_ALERT_IMMEDIATE (C macro), 41
- BLE_ANS_CMD_NOT_UNR_ALERT_IMMEDIATE (C macro), 41
- BLE_ANS_INFO_STR_MAX_LEN (C macro), 41
- BLE_ANS_NEW_ALERT_MAX_LEN (C macro), 41
- BLE_ANS_UUID16 (C macro), 40
- BLE_BAS_CHR_UUID16_BATTERY_INFO (C macro), 51
- BLE_BAS_CHR_UUID16_CRITICAL_STATUS (C macro), 51
- BLE_BAS_CHR_UUID16_ENERGY_STATUS (C macro), 51
- BLE_BAS_CHR_UUID16_ESTIMATED_SERVICE_DATA_TIME (C macro), 51
- BLE_BAS_CHR_UUID16_HEALTH_INFO (C macro), 51
- BLE_BAS_CHR_UUID16_HEALTH_STATUS (C macro), 51
- BLE_BAS_CHR_UUID16_LEVEL (C macro), 51
- BLE_BAS_CHR_UUID16_LEVEL_STATUS (C macro), 51
- BLE_BAS_CHR_UUID16_TIME_STATUS (C macro), 51
- BLE_BAS_UUID16 (C macro), 51
- BLE_BCS_CHR_UUID16_FEATURE (C macro), 57
- BLE_BCS_CHR_UUID16_MEASUREMENT (C macro), 57
- BLE_BCS_FEAT_BASAL_METABOLISM (C macro), 57
- BLE_BCS_FEAT_BODY_WATER_MASS (C macro), 57
- BLE_BCS_FEAT_FAT_FREE_MASS (C macro), 57
- BLE_BCS_FEAT_HEIGHT (C macro), 57
- BLE_BCS_FEAT_HEIGHT_RESOLUTION (C macro), 57
- BLE_BCS_FEAT_IMPEDENCE (C macro), 57
- BLE_BCS_FEAT_MASS_MEASUREMENT_RESOLUTION (C macro), 57
- BLE_BCS_FEAT_MULTI_USER (C macro), 57
- BLE_BCS_FEAT_MUSCLE_MASS (C macro), 57
- BLE_BCS_FEAT_MUSCLE_PERCENTAGE (C macro), 57
- BLE_BCS_FEAT_SOFT_LEAN_MASS (C macro), 57
- BLE_BCS_FEAT_TIME_STAMP (C macro), 57
- BLE_BCS_FEAT_WEIGHT (C macro), 57
- BLE_BCS_FLAG_BASAL_METABOLISM (C macro), 58
- BLE_BCS_FLAG_BODY_WATER_MASS (C macro), 58
- BLE_BCS_FLAG_FAT_FREE_MASS (C macro), 58
- BLE_BCS_FLAG_HEIGHT (C macro), 58
- BLE_BCS_FLAG_IMPEDENCE (C macro), 58
- BLE_BCS_FLAG_MEASUREMENT_UNITS (C macro), 57
- BLE_BCS_FLAG_MULTI_USER (C macro), 57
- BLE_BCS_FLAG_MULTIPLE_PACKET (C macro), 58
- BLE_BCS_FLAG_MUSCLE_MASS (C macro), 58
- BLE_BCS_FLAG_MUSCLE_PERCENTAGE (C macro), 58
- BLE_BCS_FLAG_SOFT_LEAN_MASS (C macro), 58
- BLE_BCS_FLAG_TIME_STAMP (C macro), 57
- BLE_BCS_FLAG_WEIGHT (C macro), 58
- BLE_BCS_MAX_VAL_LEN (C macro), 57
- BLE_BCS_UUID16 (C macro), 57
- BLE_CONN_GATT_CHR_BROADCAST (C macro), 35
- BLE_CONN_GATT_CHR_INDICATE (C macro), 36
- BLE_CONN_GATT_CHR_NOTIFY (C macro), 36
- BLE_CONN_GATT_CHR_READ (C macro), 35
- BLE_CONN_GATT_CHR_WRITE (C macro), 36
- BLE_CONN_GATT_CHR_WRITE_NO_RSP (C macro), 35
- BLE_CTS_CHANGE_OF_DST_MASK (C macro), 61
- BLE_CTS_CHANGE_OF_TIME_ZONE_MASK (C macro)

- macro*), 61
- BLE_CTS_CHR_UUID16_CURRENT_TIME (C *macro*), 61
- BLE_CTS_CHR_UUID16_LOCAL_TIME (C *macro*), 61
- BLE_CTS_CHR_UUID16_REFERENCE_TIME (C *macro*), 61
- BLE_CTS_EXTERNAL_REFERENCE_TIME_UPDATE_MASK (C *macro*), 61
- BLE_CTS_MANUAL_TIME_UPDATE_MASK (C *macro*), 61
- BLE_CTS_UUID16 (C *macro*), 61
- BLE_DIS_CHR_UUID16_FIRMWARE_REVISION (C *macro*), 65
- BLE_DIS_CHR_UUID16_HARDWARE_REVISION (C *macro*), 65
- BLE_DIS_CHR_UUID16_MANUFACTURER_NAME (C *macro*), 65
- BLE_DIS_CHR_UUID16_MODEL_NUMBER (C *macro*), 65
- BLE_DIS_CHR_UUID16_PNP_ID (C *macro*), 65
- BLE_DIS_CHR_UUID16_REG_CERT (C *macro*), 65
- BLE_DIS_CHR_UUID16_SERIAL_NUMBER (C *macro*), 65
- BLE_DIS_CHR_UUID16_SOFTWARE_REVISION (C *macro*), 65
- BLE_DIS_CHR_UUID16_SYSTEM_ID (C *macro*), 65
- BLE_DIS_UUID16 (C *macro*), 65
- ble_hci_add_to_accept_list (C++ *function*), 91
- BLE_HCI_ADDR_LEN (C *macro*), 93
- ble_hci_addr_t (C++ *type*), 93
- ble_hci_addr_type_t (C++ *enum*), 94
- ble_hci_addr_type_t::BLE_ADDR_TYPE_PUBLIC (C++ *enumerator*), 94
- ble_hci_addr_type_t::BLE_ADDR_TYPE_RANDOM (C++ *enumerator*), 94
- ble_hci_addr_type_t::BLE_ADDR_TYPE_RPA_PUBLIC (C++ *enumerator*), 94
- ble_hci_addr_type_t::BLE_ADDR_TYPE_RPA_RANDOM (C++ *enumerator*), 94
- ble_hci_adv_channel_t (C++ *enum*), 95
- ble_hci_adv_channel_t::ADV_CHNL_37 (C++ *enumerator*), 95
- ble_hci_adv_channel_t::ADV_CHNL_38 (C++ *enumerator*), 95
- ble_hci_adv_channel_t::ADV_CHNL_39 (C++ *enumerator*), 95
- ble_hci_adv_channel_t::ADV_CHNL_ALL (C++ *enumerator*), 95
- ble_hci_adv_filter_t (C++ *enum*), 95
- ble_hci_adv_filter_t::ADV_FILTER_ALLOW_SCAN_ANY_SCAN_PARAM (C++ *enumerator*), 95
- ble_hci_adv_filter_t::ADV_FILTER_ALLOW_SCAN_ANY_SCAN_PARAM_PUSH (C++ *enumerator*), 95
- ble_hci_adv_filter_t::ADV_FILTER_ALLOW_SCAN_ALL_SCAN_PARAM (C++ *enumerator*), 95
- ble_hci_adv_filter_t::ADV_FILTER_ALLOW_SCAN_ALL_SCAN_PARAM_PUSH (C++ *enumerator*), 95
- ble_hci_adv_filter_t::ADV_FILTER_ALLOW_SCAN_WLST (C++ *enumerator*), 96
- ble_hci_adv_param_t (C++ *struct*), 92
- ble_hci_adv_param_t::adv_filter_policy (C++ *member*), 92
- ble_hci_adv_param_t::adv_int_max (C++ *member*), 92
- ble_hci_adv_param_t::adv_int_min (C++ *member*), 92
- ble_hci_adv_param_t::adv_type (C++ *member*), 92
- ble_hci_adv_param_t::channel_map (C++ *member*), 92
- ble_hci_adv_param_t::own_addr_type (C++ *member*), 92
- ble_hci_adv_param_t::peer_addr (C++ *member*), 92
- ble_hci_adv_param_t::peer_addr_type (C++ *member*), 92
- ble_hci_adv_type_t (C++ *enum*), 94
- ble_hci_adv_type_t::ADV_TYPE_DIRECT_IND_HIGH (C++ *enumerator*), 95
- ble_hci_adv_type_t::ADV_TYPE_DIRECT_IND_LOW (C++ *enumerator*), 95
- ble_hci_adv_type_t::ADV_TYPE_IND (C++ *enumerator*), 95
- ble_hci_adv_type_t::ADV_TYPE_NONCONN_IND (C++ *enumerator*), 95
- ble_hci_adv_type_t::ADV_TYPE_SCAN_IND (C++ *enumerator*), 95
- ble_hci_clear_accept_list (C++ *function*), 91
- ble_hci_deinit (C++ *function*), 90
- ble_hci_dev_type_t (C++ *enum*), 94
- ble_hci_dev_type_t::BLE_HCI_DEVICE_TYPE_BLE (C++ *enumerator*), 94
- ble_hci_dev_type_t::BLE_HCI_DEVICE_TYPE_BREDR (C++ *enumerator*), 94
- ble_hci_dev_type_t::BLE_HCI_DEVICE_TYPE_DUMO (C++ *enumerator*), 94
- ble_hci_enable_meta_event (C++ *function*), 90
- ble_hci_init (C++ *function*), 90
- ble_hci_reset (C++ *function*), 90
- ble_hci_scan_cb_t (C++ *type*), 93
- ble_hci_scan_param_t (C++ *struct*), 92
- ble_hci_scan_param_t::filter_policy (C++ *member*), 93
- ble_hci_scan_param_t::own_addr_type (C++ *member*), 93
- ble_hci_scan_param_t::scan_interval (C++ *member*), 93
- ble_hci_scan_param_t::scan_type (C++ *member*), 93
- ble_hci_scan_param_t::scan_window (C++ *member*), 93
- ble_hci_scan_result_t (C++ *struct*), 91
- ble_hci_scan_result_t::adv_data_len

- (C++ member), 92
- ble_hci_scan_result_t::bda (C++ member), 91
- ble_hci_scan_result_t::ble_addr_type (C++ member), 92
- ble_hci_scan_result_t::ble_adv (C++ member), 92
- ble_hci_scan_result_t::dev_type (C++ member), 91
- ble_hci_scan_result_t::rssi (C++ member), 92
- ble_hci_scan_result_t::scan_rsp_len (C++ member), 92
- ble_hci_scan_result_t::search_evt (C++ member), 91
- ble_hci_scan_type_t (C++ enum), 95
- ble_hci_scan_type_t::BLE_SCAN_TYPE_ACTIVE (C++ enumerator), 95
- ble_hci_scan_type_t::BLE_SCAN_TYPE_PASSIVE (C++ enumerator), 95
- ble_hci_search_evt_t (C++ enum), 93
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_CHEST (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_EAR_LOBE (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_FINGER (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_FOOT (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_HAND (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_MAX (C++ enumerator), 93
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_OTHER (C++ enumerator), 94
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_RFU (C++ enumerator), 93
- ble_hci_search_evt_t::BLE_HCI_SEARCH_DISABLED_SRC_BODY_SENSOR_LOC_WRIST (C++ enumerator), 94
- ble_hci_set_adv_data (C++ function), 90
- ble_hci_set_adv_enable (C++ function), 90
- ble_hci_set_adv_param (C++ function), 90
- ble_hci_set_random_address (C++ function), 91
- ble_hci_set_register_scan_callback (C++ function), 91
- ble_hci_set_scan_enable (C++ function), 90
- ble_hci_set_scan_param (C++ function), 90
- BLE_HRP_CHR_MERSUREMENT_FLAGS_FORMAT_U16 (C macro), 85
- BLE_HRP_CHR_MERSUREMENT_FLAGS_FORMAT_U8 (C macro), 85
- BLE_HRP_CHR_MERSUREMENT_FLAGS_NOT (C macro), 85
- BLE_HRP_CHR_MERSUREMENT_FLAGS_SET (C macro), 85
- BLE_HRP_CHR_MERSUREMENT_RR_INTERVAL_MAX_NUM (C macro), 85
- BLE_HRP_CHR_UUID16_BODY_SENSOR_LOC (C macro), 85
- BLE_HRP_CHR_UUID16_HEART_RATE_CNTL_POINT (C macro), 85
- BLE_HRP_CHR_UUID16_MEASUREMENT (C macro), 85
- BLE_HRP_CMD_MAX (C macro), 86
- BLE_HRP_CMD_RESET_ENERGY_EXPENDED (C macro), 86
- BLE_HRP_CMD_RFU (C macro), 85
- BLE_HRP_FLAGS_BM_ENERGY (C macro), 85
- BLE_HRP_FLAGS_BM_FORMAT (C macro), 85
- BLE_HRP_FLAGS_BM_NONE (C macro), 85
- BLE_HRP_FLAGS_BM_RFU (C macro), 85
- BLE_HRP_FLAGS_BM_RR_INTERVAL (C macro), 85
- BLE_HRP_FLAGS_BM_SENSOR_CONTACT_DETECTED (C macro), 85
- BLE_HRP_FLAGS_BM_SENSOR_CONTACT_SUPPOTRED (C macro), 85
- BLE_HRP_UUID16 (C macro), 85
- BLE_HRS_CHR_BODY_SENSOR_LOC_CHEST (C macro), 67
- BLE_HRS_CHR_BODY_SENSOR_LOC_EAR_LOBE (C macro), 67
- BLE_HRS_CHR_BODY_SENSOR_LOC_FINGER (C macro), 67
- BLE_HRS_CHR_BODY_SENSOR_LOC_FOOT (C macro), 68
- BLE_HRS_CHR_BODY_SENSOR_LOC_HAND (C macro), 67
- BLE_HRS_CHR_BODY_SENSOR_LOC_MAX (C macro), 68
- BLE_HRS_CHR_BODY_SENSOR_LOC_OTHER (C macro), 67
- BLE_HRS_CHR_BODY_SENSOR_LOC_RFU (C macro), 68
- BLE_HRS_CHR_BODY_SENSOR_LOC_WRIST (C macro), 67
- BLE_HRS_CHR_MERSUREMENT_RR_INTERVAL_MAX_NUM (C macro), 67
- BLE_HRS_CHR_UUID16_BODY_SENSOR_LOC (C macro), 67
- BLE_HRS_CHR_UUID16_HEART_RATE_CNTL_POINT (C macro), 67
- BLE_HRS_CHR_UUID16_MEASUREMENT (C macro), 67
- BLE_HRS_UUID16 (C macro), 67
- BLE_HTP_CHR_TEMPERATURE_FLAGS_NOT (C macro), 88
- BLE_HTP_CHR_TEMPERATURE_FLAGS_SET (C macro), 88
- BLE_HTP_CHR_TEMPERATURE_TYPE_ARMPIT (C macro), 88
- BLE_HTP_CHR_TEMPERATURE_TYPE_BODY (C macro), 88
- BLE_HTP_CHR_TEMPERATURE_TYPE_EAR (C macro), 89
- BLE_HTP_CHR_TEMPERATURE_TYPE_FINGER (C macro), 89

BLE_HTP_CHR_TEMPERATURE_TYPE_GAST_TRACT (C macro), 89	BLE_HTS_CHR_TEMPERATURE_TYPE_TYMPANUM (C macro), 71
BLE_HTP_CHR_TEMPERATURE_TYPE_MAX (C macro), 89	BLE_HTS_CHR_TEMPERATURE_UNITS_CELSIUS (C macro), 71
BLE_HTP_CHR_TEMPERATURE_TYPE_MOUTH (C macro), 89	BLE_HTS_CHR_TEMPERATURE_UNITS_FAHRENHEIT (C macro), 71
BLE_HTP_CHR_TEMPERATURE_TYPE_RECTUM (C macro), 89	BLE_HTS_CHR_UUID16_INTERMEDIATE_TEMPERATURE (C macro), 70
BLE_HTP_CHR_TEMPERATURE_TYPE_RFU (C macro), 88	BLE_HTS_CHR_UUID16_MEASUREMENT_INTERVAL (C macro), 70
BLE_HTP_CHR_TEMPERATURE_TYPE_TOE (C macro), 89	BLE_HTS_CHR_UUID16_TEMPERATURE_MEASUREMENT (C macro), 70
BLE_HTP_CHR_TEMPERATURE_TYPE_TYMPANUM (C macro), 89	BLE_HTS_CHR_UUID16_TEMPERATURE_TYPE (C macro), 70
BLE_HTP_CHR_TEMPERATURE_UNITS_CELSIUS (C macro), 88	BLE_HTS_UUID16 (C macro), 70
BLE_HTP_CHR_TEMPERATURE_UNITS_FAHRENHEIT (C macro), 88	BLE_TPS_CHR_UUID16_TX_POWER_LEVEL (C macro), 72
BLE_HTP_CHR_UUID16_INTERMEDIATE_TEMPERATURE (C macro), 88	BLE_TPS_UUID16 (C macro), 72
BLE_HTP_CHR_UUID16_MEASUREMENT_INTERVAL (C macro), 88	BLE_UIDS_ATT_VAL_LEN (C macro), 75
BLE_HTP_CHR_UUID16_TEMPERATURE_MEASUREMENT (C macro), 88	BLE_UDS_CHR_UUID16_DATABASH_CHG (C macro), 75
BLE_HTP_CHR_UUID16_TEMPERATURE_TYPE (C macro), 88	BLE_UDS_CHR_UUID16_REG_USER (C macro), 75
	BLE_UDS_CHR_UUID16_USER_CTRL (C macro), 75
	BLE_UDS_CHR_UUID16_USER_INDEX (C macro), 75
BLE_HTP_FLAGS_BM_NONE (C macro), 88	BLE_UDS_USER_CTRL_CONSENT_OP (C macro), 75
BLE_HTP_FLAGS_BM_RFU (C macro), 88	
BLE_HTP_FLAGS_BM_TEMPERATURE_TYPE (C macro), 88	BLE_UDS_USER_CTRL_DEL_DATA_OP (C macro), 75
BLE_HTP_FLAGS_BM_TEMPERATURE_UNITS (C macro), 88	BLE_UDS_USER_CTRL_DEL_USER_OP (C macro), 75
BLE_HTP_FLAGS_BM_TIME_STAMP (C macro), 88	BLE_UDS_USER_CTRL_LIST_ALL_OP (C macro), 75
BLE_HTP_UUID16 (C macro), 88	
BLE_HTS_CHR_TEMPERATURE_FLAGS_NOT (C macro), 71	BLE_UDS_USER_CTRL_POINT_PARAM_LEN (C macro), 75
BLE_HTS_CHR_TEMPERATURE_FLAGS_SET (C macro), 71	BLE_UDS_USER_CTRL_REG_NEW_OP (C macro), 75
BLE_HTS_CHR_TEMPERATURE_TYPE_ARMPIT (C macro), 71	BLE_UDS_USER_CTRL_RSP_OP (C macro), 75
BLE_HTS_CHR_TEMPERATURE_TYPE_BODY (C macro), 71	BLE_UDS_UUID16 (C macro), 75
BLE_HTS_CHR_TEMPERATURE_TYPE_EAR (C macro), 71	BLE_UUID128_VAL_LEN (C macro), 35
BLE_HTS_CHR_TEMPERATURE_TYPE_FINGER (C macro), 71	BLE_UUID_CMP (C macro), 36
BLE_HTS_CHR_TEMPERATURE_TYPE_GAST_TRACT (C macro), 71	BLE_UUID_TYPE (C macro), 36
	BLE_WSS_BMI_FLAG (C macro), 79
	BLE_WSS_BMI_MASK (C macro), 78
	BLE_WSS_CHR_UUID16_WEIGHT_FEATURE (C macro), 78
	BLE_WSS_CHR_UUID16_WEIGHT_MEASUREMENT (C macro), 78
BLE_HTS_CHR_TEMPERATURE_TYPE_MAX (C macro), 71	BLE_WSS_HEIGHT_RESOLUTION_0P001_M (C macro), 79
BLE_HTS_CHR_TEMPERATURE_TYPE_MOUTH (C macro), 71	BLE_WSS_HEIGHT_RESOLUTION_0P005_M (C macro), 79
BLE_HTS_CHR_TEMPERATURE_TYPE_RECTUM (C macro), 71	BLE_WSS_HEIGHT_RESOLUTION_0P01_M (C macro), 79
BLE_HTS_CHR_TEMPERATURE_TYPE_RFU (C macro), 71	BLE_WSS_HEIGHT_RESOLUTION_NONE (C macro), 78
BLE_HTS_CHR_TEMPERATURE_TYPE_TOE (C macro), 71	BLE_WSS_HEIGHT_RESOLUTION_OFFSET (C macro), 78

- macro), 78
- BLE_WSS_MEASUREMENT_UINTS_FLAG (C macro), 79
- BLE_WSS_MULTI_USER_MASK (C macro), 78
- BLE_WSS_TIME_STAMP_FLAG (C macro), 79
- BLE_WSS_TIMESTAMP_MASK (C macro), 78
- BLE_WSS_USER_ID_FLAG (C macro), 79
- BLE_WSS_UUID16 (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P005_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P01_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P02_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P05_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P1_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P2_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_0P5_KG (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_NONE (C macro), 78
- BLE_WSS_WEIGHT_RESOLUTION_OFFSET (C macro), 78
- BROADCAST_PARAM_LEN (C macro), 35
- bus_handle_t (C++ type), 345
- button_cb_t (C++ type), 308
- button_event_args_t (C++ union), 307
- button_event_args_t::long_press (C++ member), 307
- button_event_args_t::long_press_t (C++ struct), 307
- button_event_args_t::long_press_t::press_time (C++ member), 307
- button_event_args_t::multiple_clicks (C++ member), 307
- button_event_args_t::multiple_clicks_t (C++ struct), 307
- button_event_args_t::multiple_clicks_t::clicks (C++ member), 307
- button_event_t (C++ enum), 308
- button_event_t::BUTTON_DOUBLE_CLICK (C++ enumerator), 308
- button_event_t::BUTTON_EVENT_MAX (C++ enumerator), 309
- button_event_t::BUTTON_LONG_PRESS_HOLD (C++ enumerator), 308
- button_event_t::BUTTON_LONG_PRESS_START (C++ enumerator), 308
- button_event_t::BUTTON_LONG_PRESS_UP (C++ enumerator), 308
- button_event_t::BUTTON_MULTIPLE_CLICK (C++ enumerator), 308
- button_event_t::BUTTON_NONE_PRESS (C++ enumerator), 309
- button_event_t::BUTTON_PRESS_DOWN (C++ enumerator), 308
- button_event_t::BUTTON_PRESS_END (C++ enumerator), 308
- button_event_t::BUTTON_PRESS_REPEAT (C++ enumerator), 308
- button_event_t::BUTTON_PRESS_REPEAT_DONE (C++ enumerator), 308
- button_event_t::BUTTON_PRESS_UP (C++ enumerator), 308
- button_event_t::BUTTON_SINGLE_CLICK (C++ enumerator), 308
- button_param_t (C++ enum), 309
- button_param_t::BUTTON_LONG_PRESS_TIME_MS (C++ enumerator), 309
- button_param_t::BUTTON_PARAM_MAX (C++ enumerator), 309
- button_param_t::BUTTON_SHORT_PRESS_TIME_MS (C++ enumerator), 309
- button_power_save_cb_t (C++ type), 308
- button_power_save_config_t (C++ struct), 308
- button_power_save_config_t::enter_power_save_cb (C++ member), 308
- button_power_save_config_t::usr_data (C++ member), 308
- ## C
- calibration_factor_t (C++ struct), 363
- calibration_factor_t::ki (C++ member), 363
- calibration_factor_t::kp (C++ member), 363
- calibration_factor_t::ku (C++ member), 363
- calibration_parameter_t (C++ struct), 362
- calibration_parameter_t::standard_current (C++ member), 362
- calibration_parameter_t::standard_power (C++ member), 362
- calibration_parameter_t::standard_voltage (C++ member), 362
- CDC_HOST_ANY_PID (C macro), 236
- CDC_HOST_ANY_VID (C macro), 236
- CHARGE_TIME (C macro), 412
- chip_config_t (C++ struct), 363
- chip_config_t::cf1_gpio (C++ member), 363
- chip_config_t::cf_gpio (C++ member), 363
- chip_config_t::divider_resistor (C++ member), 363
- chip_config_t::factor (C++ member), 363
- chip_config_t::pin_mode (C++ member), 363
- chip_config_t::sampling_resistor (C++ member), 363
- chip_config_t::sel_gpio (C++ member), 363
- chip_config_t::type (C++ member), 363
- control_mode_t (C++ enum), 410
- control_mode_t::BLDC_FOC (C++ enumerator), 411

control_mode_t::BLDC_SIX_STEP (C++ *enumerator*), 410

D

dac_audio_config_t (C++ *struct*), 261
 dac_audio_config_t::bits_per_sample (C++ *member*), 261
 dac_audio_config_t::dac_mode (C++ *member*), 261
 dac_audio_config_t::dma_buf_count (C++ *member*), 261
 dac_audio_config_t::dma_buf_len (C++ *member*), 262
 dac_audio_config_t::i2s_num (C++ *member*), 261
 dac_audio_config_t::max_data_size (C++ *member*), 262
 dac_audio_config_t::sample_rate (C++ *member*), 261
 dac_audio_deinit (C++ *function*), 260
 dac_audio_init (C++ *function*), 260
 dac_audio_set_param (C++ *function*), 260
 dac_audio_set_volume (C++ *function*), 261
 dac_audio_start (C++ *function*), 260
 dac_audio_stop (C++ *function*), 260
 dac_audio_write (C++ *function*), 261
 DEFAULT_AUDIO_CODEC_ADC_MONO_CFG (C *macro*), 264
 DEFAULT_AUDIO_CODEC_ADC_STEREO_CFG (C *macro*), 264
 DEFAULT_TINYUF2_NVS_CONFIG (C *macro*), 253
 DEFAULT_TINYUF2_OTA_CONFIG (C *macro*), 253
 DUTY_MAX (C *macro*), 411
 DUTY_RES (C *macro*), 411

E

ENTER_CLOSE_TIME (C *macro*), 413
 esp_bcs_val_t (C++ *struct*), 55
 esp_bcs_val_t::basal_metabolism (C++ *member*), 56
 esp_bcs_val_t::bcs_flag (C++ *member*), 56
 esp_bcs_val_t::body_water_mass (C++ *member*), 56
 esp_bcs_val_t::day (C++ *member*), 56
 esp_bcs_val_t::fat_free_mass (C++ *member*), 56
 esp_bcs_val_t::height (C++ *member*), 57
 esp_bcs_val_t::hours (C++ *member*), 56
 esp_bcs_val_t::impedance (C++ *member*), 56
 esp_bcs_val_t::minutes (C++ *member*), 56
 esp_bcs_val_t::month (C++ *member*), 56
 esp_bcs_val_t::muscle_mass (C++ *member*), 56
 esp_bcs_val_t::muscle_percentage (C++ *member*), 56
 esp_bcs_val_t::seconds (C++ *member*), 56
 esp_bcs_val_t::soft_lean_mass (C++ *member*), 56

esp_bcs_val_t::timestamp (C++ *member*), 56
 esp_bcs_val_t::user_id (C++ *member*), 56
 esp_bcs_val_t::weight (C++ *member*), 56
 esp_bcs_val_t::year (C++ *member*), 56
 ESP_BLE_ADV_DATA_LEN_MAX (C *macro*), 93
 esp_ble_anp_data_t (C++ *struct*), 81
 esp_ble_anp_data_t::cat_id (C++ *member*), 81
 esp_ble_anp_data_t::cat_info (C++ *member*), 81
 esp_ble_anp_data_t::count (C++ *member*), 81
 esp_ble_anp_data_t::new_alert_val (C++ *member*), 81
 esp_ble_anp_data_t::unr_alert_stat (C++ *member*), 81
 esp_ble_anp_data_t::[anonymous] (C++ *member*), 81
 esp_ble_anp_deinit (C++ *function*), 80
 esp_ble_anp_get_new_alert (C++ *function*), 79
 esp_ble_anp_get_unr_alert (C++ *function*), 80
 esp_ble_anp_init (C++ *function*), 80
 esp_ble_anp_option_t (C++ *enum*), 83
 esp_ble_anp_option_t::BLE_ANP_OPT_DISABLE (C++ *enumerator*), 83
 esp_ble_anp_option_t::BLE_ANP_OPT_ENABLE (C++ *enumerator*), 83
 esp_ble_anp_option_t::BLE_ANP_OPT_RECOVER (C++ *enumerator*), 83
 esp_ble_anp_set_new_alert (C++ *function*), 80
 esp_ble_anp_set_unr_alert (C++ *function*), 80
 esp_ble_ans_get_new_alert (C++ *function*), 39
 esp_ble_ans_get_unread_alert (C++ *function*), 39
 esp_ble_ans_init (C++ *function*), 40
 esp_ble_ans_set_new_alert (C++ *function*), 39
 esp_ble_ans_set_unread_alert (C++ *function*), 39
 esp_ble_bas_battery_info_t (C++ *struct*), 49
 esp_ble_bas_battery_info_t::aggregation_group (C++ *member*), 50
 esp_ble_bas_battery_info_t::chemistry (C++ *member*), 50
 esp_ble_bas_battery_info_t::critical_energy (C++ *member*), 50
 esp_ble_bas_battery_info_t::designed_capacity (C++ *member*), 50
 esp_ble_bas_battery_info_t::en_aggregation_group (C++ *member*), 49
 esp_ble_bas_battery_info_t::en_chemistry (C++ *member*), 49
 esp_ble_bas_battery_info_t::en_critical_energy

(C++ member), 49
 esp_ble_bas_battery_info_t::en_designed_capacity (C++ member), 49
 esp_ble_bas_battery_info_t::en_expiration_date (C++ member), 49
 esp_ble_bas_battery_info_t::en_low_energy (C++ member), 47
 esp_ble_bas_battery_info_t::en_manufacture_date (C++ member), 46
 esp_ble_bas_battery_info_t::en_nominal_voltage (C++ member), 47
 esp_ble_bas_battery_info_t::expiration_date (C++ member), 47
 esp_ble_bas_battery_info_t::features (C++ member), 50
 esp_ble_bas_battery_info_t::flags (C++ member), 49
 esp_ble_bas_battery_info_t::flags_reserved (C++ member), 49
 esp_ble_bas_battery_info_t::low_energy (C++ member), 50
 esp_ble_bas_battery_info_t::manufacture_date (C++ member), 46
 esp_ble_bas_battery_info_t::nominal_voltage (C++ member), 50
 esp_ble_bas_battery_info_t::rechargeable (C++ member), 49
 esp_ble_bas_battery_info_t::replaceable (C++ member), 49
 esp_ble_bas_battery_info_t::reserved (C++ member), 49
 esp_ble_bas_critical_status_t (C++ struct), 45
 esp_ble_bas_critical_status_t::critical_power (C++ member), 46
 esp_ble_bas_critical_status_t::immediate_service (C++ member), 46
 esp_ble_bas_critical_status_t::reserved (C++ member), 46
 esp_ble_bas_critical_status_t::status (C++ member), 46
 esp_ble_bas_data_t (C++ struct), 50
 esp_ble_bas_data_t::battery_info (C++ member), 51
 esp_ble_bas_data_t::battery_level (C++ member), 50
 esp_ble_bas_data_t::critical_status (C++ member), 50
 esp_ble_bas_data_t::energy_status (C++ member), 50
 esp_ble_bas_data_t::estimated_service_date (C++ member), 50
 esp_ble_bas_data_t::health_info (C++ member), 51
 esp_ble_bas_data_t::health_status (C++ member), 51
 esp_ble_bas_data_t::level_status (C++ member), 50
 esp_ble_bas_data_t::time_status (C++ member), 51
 esp_ble_bas_energy_status_t (C++ struct), 46
 esp_ble_bas_energy_status_t::available_battery_capacity (C++ member), 47
 esp_ble_bas_energy_status_t::available_energy (C++ member), 46
 esp_ble_bas_energy_status_t::available_energy_last_updated (C++ member), 46
 esp_ble_bas_energy_status_t::charge_rate (C++ member), 47
 esp_ble_bas_energy_status_t::en_available_battery_level (C++ member), 46
 esp_ble_bas_energy_status_t::en_available_energy (C++ member), 46
 esp_ble_bas_energy_status_t::en_available_energy_level (C++ member), 46
 esp_ble_bas_energy_status_t::en_charge_rate (C++ member), 46
 esp_ble_bas_energy_status_t::en_external_source_power (C++ member), 46
 esp_ble_bas_energy_status_t::en_voltage (C++ member), 46
 esp_ble_bas_energy_status_t::external_source_power (C++ member), 46
 esp_ble_bas_energy_status_t::flags (C++ member), 46
 esp_ble_bas_energy_status_t::reserved (C++ member), 46
 esp_ble_bas_energy_status_t::voltage (C++ member), 46
 esp_ble_bas_get_battery_info (C++ function), 42
 esp_ble_bas_get_battery_level (C++ function), 42
 esp_ble_bas_get_critical_status (C++ function), 42
 esp_ble_bas_get_energy_status (C++ function), 42
 esp_ble_bas_get_estimated_date (C++ function), 42
 esp_ble_bas_get_health_info (C++ function), 43
 esp_ble_bas_get_health_status (C++ function), 43
 esp_ble_bas_get_level_status (C++ function), 42
 esp_ble_bas_get_time_status (C++ function), 43
 esp_ble_bas_health_info_t (C++ struct), 48
 esp_ble_bas_health_info_t::cycle_count_designed_level (C++ member), 48
 esp_ble_bas_health_info_t::en_cycle_count_designed_level (C++ member), 48
 esp_ble_bas_health_info_t::flags (C++ member), 48
 esp_ble_bas_health_info_t::max_designed_operating

(C++ member), 49
 esp_ble_bas_health_info_t::min_designed_operating_temperature (C++ member), 48
 esp_ble_bas_health_info_t::min_max_designed_operating_temperature (C++ member), 48
 esp_ble_bas_health_info_t::reserved (C++ member), 48
 esp_ble_bas_health_status_t (C++ struct), 47
 esp_ble_bas_health_status_t::battery_health (C++ member), 48
 esp_ble_bas_health_status_t::current_temperature (C++ member), 48
 esp_ble_bas_health_status_t::cycle_count (C++ member), 48
 esp_ble_bas_health_status_t::deep_discharge_count (C++ member), 48
 esp_ble_bas_health_status_t::en_battery_health_monitoring (C++ member), 47
 esp_ble_bas_health_status_t::en_current_temperature (C++ member), 48
 esp_ble_bas_health_status_t::en_cycle_count (C++ member), 47
 esp_ble_bas_health_status_t::en_deep_discharge_count (C++ member), 48
 esp_ble_bas_health_status_t::flags (C++ member), 48
 esp_ble_bas_health_status_t::reserved (C++ member), 48
 esp_ble_bas_init (C++ function), 44
 esp_ble_bas_level_status_t (C++ struct), 44
 esp_ble_bas_level_status_t::additional_status (C++ member), 45
 esp_ble_bas_level_status_t::battery (C++ member), 44
 esp_ble_bas_level_status_t::battery_charge_level (C++ member), 45
 esp_ble_bas_level_status_t::battery_charge_status (C++ member), 45
 esp_ble_bas_level_status_t::battery_charge_type (C++ member), 45
 esp_ble_bas_level_status_t::battery_fault (C++ member), 45
 esp_ble_bas_level_status_t::battery_level (C++ member), 45
 esp_ble_bas_level_status_t::charging_fault_reason (C++ member), 45
 esp_ble_bas_level_status_t::en_additional_status (C++ member), 44
 esp_ble_bas_level_status_t::en_battery_level (C++ member), 44
 esp_ble_bas_level_status_t::en_identification (C++ member), 44
 esp_ble_bas_level_status_t::flags (C++ member), 44
 esp_ble_bas_level_status_t::flags_reserved (C++ member), 44
 esp_ble_bas_level_status_t::identifier (C++ member), 45
 esp_ble_bas_level_status_t::power_state (C++ member), 45
 esp_ble_bas_level_status_t::power_state_reserved (C++ member), 45
 esp_ble_bas_level_status_t::reserved (C++ member), 45
 esp_ble_bas_level_status_t::service_required (C++ member), 45
 esp_ble_bas_level_status_t::wired_external_power (C++ member), 45
 esp_ble_bas_level_status_t::wireless_external_power (C++ member), 45
 esp_ble_bas_set_battery_info (C++ function), 44
 esp_ble_bas_set_battery_level (C++ function), 42
 esp_ble_bas_set_critical_status (C++ function), 42
 esp_ble_bas_set_energy_status (C++ function), 43
 esp_ble_bas_set_estimated_date (C++ function), 42
 esp_ble_bas_set_health_info (C++ function), 43
 esp_ble_bas_set_health_status (C++ function), 43
 esp_ble_bas_set_level_status (C++ function), 42
 esp_ble_bas_set_time_status (C++ function), 43
 esp_ble_bas_time_status_t (C++ struct), 47
 esp_ble_bas_time_status_t::discharged (C++ member), 47
 esp_ble_bas_time_status_t::discharged_standby (C++ member), 47
 esp_ble_bas_time_status_t::en_discharged_standby (C++ member), 47
 esp_ble_bas_time_status_t::en_recharged (C++ member), 47
 esp_ble_bas_time_status_t::flags (C++ member), 47
 esp_ble_bas_time_status_t::recharged (C++ member), 47
 esp_ble_bas_time_status_t::reserved (C++ member), 47
 esp_ble_bcs_init (C++ function), 55
 esp_ble_bcs_set_measurement (C++ function), 55
 esp_ble_conn_add_svc (C++ function), 29
 esp_ble_conn_cb_t (C++ type), 36
 esp_ble_conn_character_t (C++ struct), 30
 esp_ble_conn_character_t::flag (C++ member), 30
 esp_ble_conn_character_t::name (C++ member), 30
 esp_ble_conn_character_t::type (C++ member), 30

esp_ble_conn_character_t::uuid (C++ member), 30
 esp_ble_conn_character_t::uuid_fn (C++ member), 30
 esp_ble_conn_config_t (C++ struct), 31
 esp_ble_conn_config_t::adv_uuid16 (C++ member), 31
 esp_ble_conn_config_t::broadcast_data (C++ member), 31
 esp_ble_conn_config_t::device_name (C++ member), 31
 esp_ble_conn_config_t::extended_adv_data (C++ member), 31
 esp_ble_conn_config_t::extended_adv_len (C++ member), 31
 esp_ble_conn_config_t::extended_adv_rsp_data (C++ member), 31
 esp_ble_conn_config_t::extended_adv_rsp_len (C++ member), 31
 esp_ble_conn_config_t::include_service_data (C++ member), 31
 esp_ble_conn_config_t::periodic_adv_data (C++ member), 31
 esp_ble_conn_config_t::periodic_adv_len (C++ member), 31
 esp_ble_conn_connect (C++ function), 28
 esp_ble_conn_data_t (C++ struct), 31
 esp_ble_conn_data_t::data (C++ member), 32
 esp_ble_conn_data_t::data_len (C++ member), 32
 esp_ble_conn_data_t::type (C++ member), 31
 esp_ble_conn_data_t::uuid (C++ member), 32
 esp_ble_conn_data_t::write_conn_id (C++ member), 31
 esp_ble_conn_deinit (C++ function), 28
 esp_ble_conn_desc_t (C++ enum), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASC_FORMAT (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASYNC_CONN (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASYNC_CONN_BLOCK (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASYNC_CONN_BLOCK_ASYNC (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASYNC_CONN_BLOCK_ASYNC_MEASURE (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_ASYNC_CONN_BLOCK_ASYNC_MEASURE_TRIGGER (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_EXTENDED (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_EXTENDED_REASON (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_EXTENDED_REASON_FORMAT (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_REPORT (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_SERVER_CONN (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_TIME_TRIGGER (C++ enumerator), 38
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_UNKNOWN (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_USER (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_VALID_RANGE (C++ enumerator), 37
 esp_ble_conn_desc_t::ESP_BLE_CONN_DESC_VALUE_TRIGGER (C++ enumerator), 38
 esp_ble_conn_disconnect (C++ function), 28
 esp_ble_conn_event_t (C++ enum), 36
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_CONNECTED (C++ enumerator), 36
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_DATA_RECV (C++ enumerator), 37
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_DISCONN (C++ enumerator), 37
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_DISCONNECTED (C++ enumerator), 37
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_PERIODIC (C++ enumerator), 37
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_PERIODIC_STOPPED (C++ enumerator), 37
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_STARTED (C++ enumerator), 36
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_STOPPED (C++ enumerator), 36
 esp_ble_conn_event_t::ESP_BLE_CONN_EVENT_UNKNOWN (C++ enumerator), 36
 esp_ble_conn_init (C++ function), 27
 esp_ble_conn_notify (C++ function), 28
 esp_ble_conn_periodic_report_t (C++ struct), 32
 esp_ble_conn_periodic_report_t::data (C++ member), 33
 esp_ble_conn_periodic_report_t::data_length (C++ member), 33
 esp_ble_conn_periodic_report_t::data_status (C++ member), 33
 esp_ble_conn_periodic_report_t::rssi (C++ member), 33
 esp_ble_conn_periodic_report_t::sync_handle (C++ member), 32
 esp_ble_conn_periodic_report_t::tx_power (C++ member), 32
 esp_ble_conn_periodic_sync_lost_t (C++ struct), 33
 esp_ble_conn_periodic_sync_lost_t::reason (C++ member), 33
 esp_ble_conn_periodic_sync_lost_t::sync_handle (C++ member), 33

(C++ member), 33

esp_ble_conn_periodic_sync_t (C++ struct), 32

esp_ble_conn_periodic_sync_t::adv_addr (C++ member), 32

esp_ble_conn_periodic_sync_t::adv_clk_accuracy (C++ member), 32

esp_ble_conn_periodic_sync_t::adv_phy (C++ member), 32

esp_ble_conn_periodic_sync_t::per_adv_ival (C++ member), 32

esp_ble_conn_periodic_sync_t::sid (C++ member), 32

esp_ble_conn_periodic_sync_t::status (C++ member), 32

esp_ble_conn_periodic_sync_t::sync_handler (C++ member), 32

esp_ble_conn_read (C++ function), 29

esp_ble_conn_remove_svc (C++ function), 29

esp_ble_conn_set_mtu (C++ function), 28

esp_ble_conn_start (C++ function), 28

esp_ble_conn_stop (C++ function), 28

esp_ble_conn_subscribe (C++ function), 29

esp_ble_conn_svc_t (C++ struct), 30

esp_ble_conn_svc_t::nu_lookup (C++ member), 30

esp_ble_conn_svc_t::nu_lookup_count (C++ member), 30

esp_ble_conn_svc_t::type (C++ member), 30

esp_ble_conn_svc_t::uuid (C++ member), 30

esp_ble_conn_uuid_t (C++ union), 29

esp_ble_conn_uuid_t::uuid128 (C++ member), 30

esp_ble_conn_uuid_t::uuid16 (C++ member), 29

esp_ble_conn_uuid_t::uuid32 (C++ member), 30

esp_ble_conn_uuid_type_t (C++ enum), 38

esp_ble_conn_uuid_type_t::BLE_CONN_UUID_TYPE_128 (C++ enumerator), 38

esp_ble_conn_uuid_type_t::BLE_CONN_UUID_TYPE_16 (C++ enumerator), 38

esp_ble_conn_uuid_type_t::BLE_CONN_UUID_TYPE_32 (C++ enumerator), 38

esp_ble_conn_write (C++ function), 29

esp_ble_cts_cur_time_t (C++ struct), 59

esp_ble_cts_cur_time_t::adjust_reason (C++ member), 60

esp_ble_cts_cur_time_t::day (C++ member), 60

esp_ble_cts_cur_time_t::day_of_week (C++ member), 60

esp_ble_cts_cur_time_t::fractions_256 (C++ member), 60

esp_ble_cts_cur_time_t::hours (C++ member), 60

esp_ble_cts_cur_time_t::minutes (C++ member), 60

esp_ble_cts_cur_time_t::month (C++ member), 59

esp_ble_cts_cur_time_t::seconds (C++ member), 60

esp_ble_cts_cur_time_t::timestamp (C++ member), 60

esp_ble_cts_cur_time_t::year (C++ member), 59

esp_ble_cts_get_current_time (C++ function), 58

esp_ble_cts_get_local_time (C++ function), 59

esp_ble_cts_get_reference_time (C++ function), 59

esp_ble_cts_init (C++ function), 59

esp_ble_cts_local_time_t (C++ struct), 60

esp_ble_cts_local_time_t::dst_offset (C++ member), 60

esp_ble_cts_local_time_t::timezone (C++ member), 60

esp_ble_cts_ref_time_t (C++ struct), 60

esp_ble_cts_ref_time_t::days_since_update (C++ member), 60

esp_ble_cts_ref_time_t::hours_since_update (C++ member), 61

esp_ble_cts_ref_time_t::time_accuracy (C++ member), 60

esp_ble_cts_ref_time_t::time_source (C++ member), 60

esp_ble_cts_set_current_time (C++ function), 58

esp_ble_cts_set_local_time (C++ function), 59

esp_ble_cts_set_reference_time (C++ function), 59

esp_ble_dis_data (C++ struct), 64

esp_ble_dis_data::firmware_revision (C++ member), 64

esp_ble_dis_data::hardware_revision (C++ member), 64

esp_ble_dis_data::manufacturer_name (C++ member), 64

esp_ble_dis_data::model_number (C++ member), 64

esp_ble_dis_data::pnp_id (C++ member), 64

esp_ble_dis_data::serial_number (C++ member), 64

esp_ble_dis_data::software_revision (C++ member), 64

esp_ble_dis_data::system_id (C++ member), 64

esp_ble_dis_data_t (C++ type), 65

esp_ble_dis_get_firmware_revision (C++ function), 62

esp_ble_dis_get_hardware_revision (C++ function), 62

esp_ble_dis_get_manufacturer_name (C++ function), 63

esp_ble_dis_get_model_number (C++ function), 61
 esp_ble_dis_get_pnp_id (C++ function), 63
 esp_ble_dis_get_serial_number (C++ function), 62
 esp_ble_dis_get_software_revision (C++ function), 62
 esp_ble_dis_get_system_id (C++ function), 63
 esp_ble_dis_init (C++ function), 63
 esp_ble_dis_pnp (C++ struct), 64
 esp_ble_dis_pnp::pid (C++ member), 64
 esp_ble_dis_pnp::src (C++ member), 64
 esp_ble_dis_pnp::ver (C++ member), 64
 esp_ble_dis_pnp::vid (C++ member), 64
 esp_ble_dis_pnp_t (C++ type), 65
 esp_ble_dis_set_firmware_revision (C++ function), 62
 esp_ble_dis_set_hardware_revision (C++ function), 62
 esp_ble_dis_set_manufacturer_name (C++ function), 63
 esp_ble_dis_set_model_number (C++ function), 61
 esp_ble_dis_set_pnp_id (C++ function), 63
 esp_ble_dis_set_serial_number (C++ function), 62
 esp_ble_dis_set_software_revision (C++ function), 62
 esp_ble_dis_set_system_id (C++ function), 63
 esp_ble_hrp_data_t (C++ struct), 84
 esp_ble_hrp_data_t::detected (C++ member), 84
 esp_ble_hrp_data_t::energy (C++ member), 84
 esp_ble_hrp_data_t::energy_val (C++ member), 85
 esp_ble_hrp_data_t::flags (C++ member), 84
 esp_ble_hrp_data_t::format (C++ member), 84
 esp_ble_hrp_data_t::heartrate (C++ member), 85
 esp_ble_hrp_data_t::interval (C++ member), 84
 esp_ble_hrp_data_t::interval_buf (C++ member), 85
 esp_ble_hrp_data_t::reserved (C++ member), 84
 esp_ble_hrp_data_t::supported (C++ member), 84
 esp_ble_hrp_data_t::u16 (C++ member), 84
 esp_ble_hrp_data_t::u8 (C++ member), 84
 esp_ble_hrp_deinit (C++ function), 84
 esp_ble_hrp_get_ctrl (C++ function), 83
 esp_ble_hrp_get_location (C++ function), 83
 esp_ble_hrp_init (C++ function), 84
 esp_ble_hrp_set_ctrl (C++ function), 83
 esp_ble_hrs_get_hrm (C++ function), 66
 esp_ble_hrs_get_location (C++ function), 66
 esp_ble_hrs_hrm_t (C++ struct), 66
 esp_ble_hrs_hrm_t::detected (C++ member), 66
 esp_ble_hrs_hrm_t::energy (C++ member), 66
 esp_ble_hrs_hrm_t::energy_val (C++ member), 67
 esp_ble_hrs_hrm_t::flags (C++ member), 67
 esp_ble_hrs_hrm_t::format (C++ member), 66
 esp_ble_hrs_hrm_t::heartrate (C++ member), 67
 esp_ble_hrs_hrm_t::interval (C++ member), 67
 esp_ble_hrs_hrm_t::interval_buf (C++ member), 67
 esp_ble_hrs_hrm_t::reserved (C++ member), 67
 esp_ble_hrs_hrm_t::supported (C++ member), 66
 esp_ble_hrs_hrm_t::u16 (C++ member), 67
 esp_ble_hrs_hrm_t::u8 (C++ member), 67
 esp_ble_hrs_init (C++ function), 66
 esp_ble_hrs_set_hrm (C++ function), 66
 esp_ble_hrs_set_location (C++ function), 66
 esp_ble_http_data_t (C++ struct), 87
 esp_ble_http_data_t::celsius (C++ member), 87
 esp_ble_http_data_t::day (C++ member), 87
 esp_ble_http_data_t::fahrenheit (C++ member), 87
 esp_ble_http_data_t::flags (C++ member), 87
 esp_ble_http_data_t::hours (C++ member), 87
 esp_ble_http_data_t::location (C++ member), 88
 esp_ble_http_data_t::minutes (C++ member), 87
 esp_ble_http_data_t::month (C++ member), 87
 esp_ble_http_data_t::reserved (C++ member), 87
 esp_ble_http_data_t::seconds (C++ member), 88
 esp_ble_http_data_t::temperature (C++ member), 87
 esp_ble_http_data_t::temperature_type (C++ member), 87
 esp_ble_http_data_t::temperature_unit (C++ member), 87
 esp_ble_http_data_t::time_stamp (C++ member), 87
 esp_ble_http_data_t::timestamp (C++ member), 88

- esp_ble_htp_data_t::year (C++ member), 87
 esp_ble_htp_deinit (C++ function), 86
 esp_ble_htp_get_measurement_interval (C++ function), 86
 esp_ble_htp_get_temp_type (C++ function), 86
 esp_ble_htp_init (C++ function), 86
 esp_ble_htp_set_measurement_interval (C++ function), 86
 esp_ble_hsts_get_intermediate_temp (C++ function), 68
 esp_ble_hsts_get_measurement_interval (C++ function), 69
 esp_ble_hsts_get_measurement_temp (C++ function), 68
 esp_ble_hsts_get_temp_type (C++ function), 68
 esp_ble_hsts_init (C++ function), 69
 esp_ble_hsts_set_intermediate_temp (C++ function), 69
 esp_ble_hsts_set_measurement_interval (C++ function), 69
 esp_ble_hsts_set_measurement_temp (C++ function), 68
 esp_ble_hsts_set_temp_type (C++ function), 68
 esp_ble_hsts_temp_t (C++ struct), 69
 esp_ble_hsts_temp_t::celsius (C++ member), 70
 esp_ble_hsts_temp_t::day (C++ member), 70
 esp_ble_hsts_temp_t::fahrenheit (C++ member), 70
 esp_ble_hsts_temp_t::flags (C++ member), 70
 esp_ble_hsts_temp_t::hours (C++ member), 70
 esp_ble_hsts_temp_t::location (C++ member), 70
 esp_ble_hsts_temp_t::minutes (C++ member), 70
 esp_ble_hsts_temp_t::month (C++ member), 70
 esp_ble_hsts_temp_t::reserved (C++ member), 69
 esp_ble_hsts_temp_t::seconds (C++ member), 70
 esp_ble_hsts_temp_t::temperature (C++ member), 70
 esp_ble_hsts_temp_t::temperature_type (C++ member), 69
 esp_ble_hsts_temp_t::temperature_unit (C++ member), 69
 esp_ble_hsts_temp_t::time_stamp (C++ member), 69
 esp_ble_hsts_temp_t::timestamp (C++ member), 70
 esp_ble_hsts_temp_t::year (C++ member), 70
 ESP_BLE_SCAN_RSP_DATA_LEN_MAX (C macro), 93
 esp_ble_tps_get_tx_power_level (C++ function), 72
 esp_ble_tps_init (C++ function), 72
 esp_ble_tps_set_tx_power_level (C++ function), 72
 esp_ble_uds_db_t (C++ struct), 73
 esp_ble_uds_db_t::db_buf (C++ member), 74
 esp_ble_uds_db_t::len (C++ member), 73
 esp_ble_uds_get_db_incre (C++ function), 72
 esp_ble_uds_get_reg_user (C++ function), 73
 esp_ble_uds_get_user_ctrl (C++ function), 73
 esp_ble_uds_init (C++ function), 73
 esp_ble_uds_reg_user_t (C++ struct), 74
 esp_ble_uds_reg_user_t::first_segment (C++ member), 74
 esp_ble_uds_reg_user_t::flag (C++ member), 74
 esp_ble_uds_reg_user_t::header (C++ member), 74
 esp_ble_uds_reg_user_t::last_segment (C++ member), 74
 esp_ble_uds_reg_user_t::rolling_segment_number (C++ member), 74
 esp_ble_uds_reg_user_t::user_index (C++ member), 74
 esp_ble_uds_reg_user_t::user_name (C++ member), 74
 esp_ble_uds_reg_user_t::user_name_present (C++ member), 74
 esp_ble_uds_reg_user_t::user_name_truncated (C++ member), 74
 esp_ble_uds_set_db_incre (C++ function), 72
 esp_ble_uds_set_reg_user (C++ function), 73
 esp_ble_uds_set_user_ctrl (C++ function), 73
 esp_ble_uds_user_ctrl_t (C++ struct), 74
 esp_ble_uds_user_ctrl_t::op_code (C++ member), 74
 esp_ble_uds_user_ctrl_t::param_len (C++ member), 74
 esp_ble_uds_user_ctrl_t::parameter (C++ member), 74
 esp_ble_wss_feature_t (C++ struct), 76
 esp_ble_wss_feature_t::bmi (C++ member), 76
 esp_ble_wss_feature_t::h_resolution (C++ member), 76
 esp_ble_wss_feature_t::height (C++ member), 76
 esp_ble_wss_feature_t::timestamp (C++ member), 76
 esp_ble_wss_feature_t::user_id (C++ member), 76
 esp_ble_wss_feature_t::w_resolution (C++ member), 76
 esp_ble_wss_feature_t::weight (C++

- member*), 76
- esp_ble_wss_get_measurement (C++ *function*), 76
- esp_ble_wss_init (C++ *function*), 76
- esp_ble_wss_measurement_t (C++ *struct*), 76
- esp_ble_wss_measurement_t::bmi (C++ *member*), 77
- esp_ble_wss_measurement_t::bmi_height (C++ *member*), 77
- esp_ble_wss_measurement_t::day (C++ *member*), 77
- esp_ble_wss_measurement_t::flag (C++ *member*), 77
- esp_ble_wss_measurement_t::height (C++ *member*), 78
- esp_ble_wss_measurement_t::height_resolution (C++ *member*), 78
- esp_ble_wss_measurement_t::hours (C++ *member*), 77
- esp_ble_wss_measurement_t::measurement (C++ *member*), 77
- esp_ble_wss_measurement_t::minutes (C++ *member*), 77
- esp_ble_wss_measurement_t::month (C++ *member*), 77
- esp_ble_wss_measurement_t::seconds (C++ *member*), 77
- esp_ble_wss_measurement_t::time_present (C++ *member*), 77
- esp_ble_wss_measurement_t::timestamp (C++ *member*), 77
- esp_ble_wss_measurement_t::user_id (C++ *member*), 77
- esp_ble_wss_measurement_t::user_present (C++ *member*), 77
- esp_ble_wss_measurement_t::weight (C++ *member*), 77
- esp_ble_wss_measurement_t::weight_resolution (C++ *member*), 78
- esp_ble_wss_measurement_t::year (C++ *member*), 77
- esp_ble_wss_set_measurement (C++ *function*), 76
- ESP_EVENT_DECLARE_BASE (C++ *function*), 408
- ESP_IOT_ATT_ALREADY_OPEN (C *macro*), 35
- ESP_IOT_ATT_APP_RSP (C *macro*), 35
- ESP_IOT_ATT_AUTH_FAIL (C *macro*), 34
- ESP_IOT_ATT_BUSY (C *macro*), 34
- ESP_IOT_ATT_CANCEL (C *macro*), 35
- ESP_IOT_ATT_CCC_CFG_ERR (C *macro*), 35
- ESP_IOT_ATT_CMD_STARTED (C *macro*), 34
- ESP_IOT_ATT_CONGESTED (C *macro*), 35
- ESP_IOT_ATT_DATABASE_OUT_OF_SYNC (C *macro*), 34
- ESP_IOT_ATT_DB_FULL (C *macro*), 34
- ESP_IOT_ATT_DUP_REG (C *macro*), 35
- ESP_IOT_ATT_ENCRYPED_MITM (C *macro*), 35
- ESP_IOT_ATT_ENCRYPED_NO_MITM (C *macro*), 35
- ESP_IOT_ATT_ERR_UNLIKELY (C *macro*), 34
- ESP_IOT_ATT_ERROR (C *macro*), 34
- ESP_IOT_ATT_ILLEGAL_PARAMETER (C *macro*), 34
- ESP_IOT_ATT_INSUF_AUTHENTICATION (C *macro*), 33
- ESP_IOT_ATT_INSUF_AUTHORIZATION (C *macro*), 33
- ESP_IOT_ATT_INSUF_ENCRYPTION (C *macro*), 34
- ESP_IOT_ATT_INSUF_KEY_SIZE (C *macro*), 34
- ESP_IOT_ATT_INSUF_RESOURCE (C *macro*), 34
- ESP_IOT_ATT_INTERNAL_ERROR (C *macro*), 34
- ESP_IOT_ATT_INVALID_ATTR_LEN (C *macro*), 34
- ESP_IOT_ATT_INVALID_CFG (C *macro*), 34
- ESP_IOT_ATT_INVALID_HANDLE (C *macro*), 33
- ESP_IOT_ATT_INVALID_OFFSET (C *macro*), 33
- ESP_IOT_ATT_INVALID_PDU (C *macro*), 33
- ESP_IOT_ATT_MORE (C *macro*), 34
- ESP_IOT_ATT_NO_RESOURCES (C *macro*), 34
- ESP_IOT_ATT_NOT_ENCRYPTED (C *macro*), 35
- ESP_IOT_ATT_NOT_FOUND (C *macro*), 34
- ESP_IOT_ATT_NOT_LONG (C *macro*), 34
- ESP_IOT_ATT_OUT_OF_RANGE (C *macro*), 35
- ESP_IOT_ATT_PENDING (C *macro*), 34
- ESP_IOT_ATT_PRC_IN_PROGRESS (C *macro*), 35
- ESP_IOT_ATT_PREPARE_Q_FULL (C *macro*), 33
- ESP_IOT_ATT_READ_NOT_PERMIT (C *macro*), 33
- ESP_IOT_ATT_REQ_NOT_SUPPORTED (C *macro*), 33
- ESP_IOT_ATT_SERVICE_STARTED (C *macro*), 35
- ESP_IOT_ATT_STACK_RSP (C *macro*), 35
- ESP_IOT_ATT_SUCCESS (C *macro*), 33
- ESP_IOT_ATT_UNKNOWN_ERROR (C *macro*), 35
- ESP_IOT_ATT_UNSUPPORT_GRP_TYPE (C *macro*), 34
- ESP_IOT_ATT_VALUE_NOT_ALLOWED (C *macro*), 34
- ESP_IOT_ATT_WRITE_NOT_PERMIT (C *macro*), 33
- ESP_IOT_ATT_WRONG_STATE (C *macro*), 34
- esp_lv_decoder_deinit (C++ *function*), 173
- esp_lv_decoder_handle_t (C++ *type*), 174
- esp_lv_decoder_init (C++ *function*), 173
- esp_lv_fs_desc_deinit (C++ *function*), 175
- esp_lv_fs_desc_init (C++ *function*), 175
- esp_lv_fs_handle_t (C++ *type*), 175
- esp_msc_ota (C++ *function*), 226
- esp_msc_ota_abort (C++ *function*), 226
- esp_msc_ota_begin (C++ *function*), 225
- esp_msc_ota_config_t (C++ *struct*), 227
- esp_msc_ota_config_t::buffer_size (C++ *member*), 227
- esp_msc_ota_config_t::bulk_flash_erase (C++ *member*), 227

- esp_msc_ota_config_t::ota_bin_path (C++ member), 227
 esp_msc_ota_config_t::skip_msc_connect (C++ member), 227
 esp_msc_ota_config_t::wait_msc_connect (C++ member), 227
 esp_msc_ota_end (C++ function), 226
 esp_msc_ota_event_t (C++ enum), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_ABORT (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_FAILED (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_FINISH (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_GET_IMG_DESC (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_READY (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_START (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_UPDATE_BOOT_PARTITION (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_VERIFY_CHIP_ID (C++ enumerator), 228
 esp_msc_ota_event_t::ESP_MSC_OTA_WRITE_FS (C++ enumerator), 228
 esp_msc_ota_get_img_desc (C++ function), 226
 esp_msc_ota_get_status (C++ function), 227
 esp_msc_ota_handle_t (C++ type), 227
 esp_msc_ota_is_complete_data_received (C++ function), 227
 esp_msc_ota_perform (C++ function), 225
 esp_msc_ota_set_msc_connect_state (C++ function), 226
 esp_msc_ota_status_t (C++ enum), 228
 esp_msc_ota_status_t::ESP_MSC_OTA_BEGIN (C++ enumerator), 228
 esp_msc_ota_status_t::ESP_MSC_OTA_IN_PROGRESS (C++ enumerator), 228
 esp_msc_ota_status_t::ESP_MSC_OTA_INIT (C++ enumerator), 228
 esp_msc_ota_status_t::ESP_MSC_OTA_SUCCESS (C++ enumerator), 228
 esp_restart_from_tinyuf2 (C++ function), 252
 esp_tinyuf2_current_state (C++ function), 252
 esp_tinyuf2_install (C++ function), 252
 esp_tinyuf2_uninstall (C++ function), 252
- ## F
- FLAG_UAC_MIC_SUSPEND_AFTER_START (C macro), 222
 FLAG_UAC_SPK_SUSPEND_AFTER_START (C macro), 222
 FLAG_UVC_SUSPEND_AFTER_START (C macro), 222
- FPS2INTERVAL (C macro), 222
 frame_data_t (C++ struct), 267
 frame_data_t::audio_info (C++ member), 268
 frame_data_t::data (C++ member), 267
 frame_data_t::data_bytes (C++ member), 267
 frame_data_t::type (C++ member), 268
 frame_data_t::video_info (C++ member), 268
 frame_data_t::[anonymous] (C++ member), 268
 FRAME_INTERVAL_FPS_10 (C macro), 222
 FRAME_INTERVAL_FPS_15 (C macro), 222
 FRAME_INTERVAL_FPS_20 (C macro), 222
 FRAME_INTERVAL_FPS_30 (C macro), 222
 FRAME_INTERVAL_FPS_5 (C macro), 222
 FRAME_RESOLUTION_ANY (C macro), 222
 frame_type_t (C++ enum), 269
 frame_type_t::FRAME_TYPE_AUDIO (C++ member), 269
 frame_type_t::FRAME_TYPE_VIDEO (C++ member), 269
 FS (C macro), 269
 fs_cfg_t (C++ struct), 175
 fs_cfg_t::fs_assets (C++ member), 175
 fs_cfg_t::fs_letter (C++ member), 175
 fs_cfg_t::fs_nums (C++ member), 175
- ## G
- gpio_pad_select_gpio (C macro), 14
- ## I
- i2c_bus_cmd_begin (C++ function), 12
 i2c_bus_create (C++ function), 8
 i2c_bus_delete (C++ function), 8
 i2c_bus_device_create (C++ function), 9
 i2c_bus_device_delete (C++ function), 9
 i2c_bus_device_get_address (C++ function), 9
 i2c_bus_device_handle_t (C++ type), 14
 i2c_bus_get_created_device_num (C++ function), 9
 i2c_bus_get_current_clk_speed (C++ function), 9
 i2c_bus_get_internal_bus_handle (C++ function), 9
 i2c_bus_handle_t (C++ type), 14
 i2c_bus_read_bit (C++ function), 10
 i2c_bus_read_bits (C++ function), 10
 i2c_bus_read_byte (C++ function), 9
 i2c_bus_read_bytes (C++ function), 10
 i2c_bus_read_reg16 (C++ function), 12
 i2c_bus_scan (C++ function), 9
 i2c_bus_write_bit (C++ function), 11
 i2c_bus_write_bits (C++ function), 11
 i2c_bus_write_byte (C++ function), 11
 i2c_bus_write_bytes (C++ function), 11

- i2c_bus_write_reg16 (C++ function), 12
- i2c_cmd_handle_t (C++ type), 14
- i2c_config_t (C++ struct), 13
- i2c_config_t::clk_flags (C++ member), 13
- i2c_config_t::clk_speed (C++ member), 13
- i2c_config_t::master (C++ member), 13
- i2c_config_t::mode (C++ member), 13
- i2c_config_t::scl_io_num (C++ member), 13
- i2c_config_t::scl_pullup_en (C++ member), 13
- i2c_config_t::sda_io_num (C++ member), 13
- i2c_config_t::sda_pullup_en (C++ member), 13
- i2s_lcd_acquire (C++ function), 20
- i2s_lcd_config_t (C++ struct), 20
- i2s_lcd_config_t::buffer_size (C++ member), 21
- i2s_lcd_config_t::clk_freq (C++ member), 20
- i2s_lcd_config_t::data_width (C++ member), 20
- i2s_lcd_config_t::i2s_port (C++ member), 21
- i2s_lcd_config_t::pin_data_num (C++ member), 20
- i2s_lcd_config_t::pin_num_cs (C++ member), 20
- i2s_lcd_config_t::pin_num_rs (C++ member), 20
- i2s_lcd_config_t::pin_num_wr (C++ member), 20
- i2s_lcd_config_t::swap_data (C++ member), 21
- i2s_lcd_driver_deinit (C++ function), 19
- i2s_lcd_driver_init (C++ function), 19
- i2s_lcd_handle_t (C++ type), 21
- i2s_lcd_release (C++ function), 20
- i2s_lcd_write (C++ function), 20
- i2s_lcd_write_cmd (C++ function), 19
- i2s_lcd_write_command (C++ function), 19
- i2s_lcd_write_data (C++ function), 19
- ina236_clear_mask (C++ function), 358
- ina236_config_t (C++ struct), 358
- ina236_config_t::alert_cb (C++ member), 359
- ina236_config_t::alert_en (C++ member), 359
- ina236_config_t::alert_pin (C++ member), 359
- ina236_config_t::bus (C++ member), 359
- ina236_config_t::dev_addr (C++ member), 359
- ina236_create (C++ function), 358
- ina236_delete (C++ function), 358
- ina236_get_current (C++ function), 358
- ina236_get_voltage (C++ function), 358
- ina236_handle_t (C++ type), 359
- INA236_I2C_ADDRESS_DEFAULT (C macro), 359
- INJECT_DUTY (C macro), 412
- INJECT_ENABLE (C macro), 412
- int236_alert_cb_t (C++ type), 359
- iot_button_count_cb (C++ function), 305
- iot_button_count_event_cb (C++ function), 305
- iot_button_delete (C++ function), 305
- iot_button_get_event (C++ function), 305
- iot_button_get_event_str (C++ function), 306
- iot_button_get_key_level (C++ function), 306
- iot_button_get_long_press_hold_cnt (C++ function), 306
- iot_button_get_repeat (C++ function), 306
- iot_button_get_ticks_time (C++ function), 306
- iot_button_print_event (C++ function), 306
- iot_button_register_cb (C++ function), 305
- iot_button_register_power_save_cb (C++ function), 307
- iot_button_resume (C++ function), 306
- iot_button_set_param (C++ function), 306
- iot_button_stop (C++ function), 307
- iot_button_unregister_cb (C++ function), 305
- iot_eth_config_t (C++ struct), 458
- iot_eth_config_t::driver (C++ member), 459
- iot_eth_config_t::stack_input (C++ member), 459
- iot_eth_config_t::user_data (C++ member), 459
- iot_eth_event_t (C++ enum), 459
- iot_eth_event_t::IOT_ETH_EVENT_CONNECTED (C++ enumerator), 459
- iot_eth_event_t::IOT_ETH_EVENT_DISCONNECTED (C++ enumerator), 459
- iot_eth_event_t::IOT_ETH_EVENT_START (C++ enumerator), 459
- iot_eth_event_t::IOT_ETH_EVENT_STOP (C++ enumerator), 459
- iot_eth_get_addr (C++ function), 458
- iot_eth_handle_t (C++ type), 459
- iot_eth_install (C++ function), 458
- iot_eth_new_usb_ecm (C++ function), 243
- iot_eth_new_usb_rndis (C++ function), 240
- iot_eth_start (C++ function), 458
- iot_eth_stop (C++ function), 458
- iot_eth_transmit (C++ function), 458
- iot_eth_uninstall (C++ function), 458
- iot_eth_update_input_path (C++ function), 458
- iot_knob_clear_count_value (C++ function), 320
- iot_knob_create (C++ function), 320
- iot_knob_delete (C++ function), 320
- iot_knob_get_count_value (C++ function),

- 320
- iot_knob_get_event (C++ function), 320
- iot_knob_register_cb (C++ function), 320
- iot_knob_resume (C++ function), 320
- iot_knob_stop (C++ function), 321
- iot_knob_unregister_cb (C++ function), 320
- iot_sensor_create (C++ function), 349
- iot_sensor_delete (C++ function), 349
- iot_sensor_handler_register (C++ function), 349
- iot_sensor_handler_register_with_type (C++ function), 350
- iot_sensor_handler_unregister (C++ function), 350
- iot_sensor_handler_unregister_with_type (C++ function), 350
- iot_sensor_scan (C++ function), 349
- iot_sensor_start (C++ function), 349
- iot_sensor_stop (C++ function), 349
- iot_servo_deinit (C++ function), 416
- iot_servo_init (C++ function), 415
- iot_servo_read_angle (C++ function), 416
- iot_servo_write_angle (C++ function), 416
- iot_usbh_ecm_config_t (C++ struct), 243
- iot_usbh_ecm_config_t::auto_detect (C++ member), 243
- iot_usbh_ecm_config_t::auto_detect_timeout (C++ member), 243
- iot_usbh_ecm_config_t::itf_num (C++ member), 243
- iot_usbh_ecm_config_t::pid (C++ member), 243
- iot_usbh_ecm_config_t::vid (C++ member), 243
- iot_usbh_rndis_config_t (C++ struct), 240
- iot_usbh_rndis_config_t::auto_detect (C++ member), 241
- iot_usbh_rndis_config_t::auto_detect_timeout (C++ member), 241
- iot_usbh_rndis_config_t::itf_num (C++ member), 241
- iot_usbh_rndis_config_t::pid (C++ member), 241
- iot_usbh_rndis_config_t::vid (C++ member), 241
- ir_learn_add_list_node (C++ function), 332
- ir_learn_add_sub_list_node (C++ function), 332
- ir_learn_cfg_t (C++ struct), 333
- ir_learn_cfg_t::callback (C++ member), 333
- ir_learn_cfg_t::clk_src (C++ member), 333
- ir_learn_cfg_t::learn_count (C++ member), 333
- ir_learn_cfg_t::learn_gpio (C++ member), 333
- ir_learn_cfg_t::resolution (C++ member), 333
- ir_learn_cfg_t::task_affinity (C++ member), 334
- ir_learn_cfg_t::task_priority (C++ member), 333
- ir_learn_cfg_t::task_stack (C++ member), 334
- ir_learn_check_valid (C++ function), 332
- ir_learn_clean_data (C++ function), 332
- ir_learn_clean_sub_data (C++ function), 332
- ir_learn_handle_t (C++ type), 334
- ir_learn_list_t (C++ struct), 333
- ir_learn_list_t::cmd_sub_node (C++ member), 333
- ir_learn_new (C++ function), 331
- ir_learn_print_raw (C++ function), 332
- ir_learn_restart (C++ function), 331
- ir_learn_result_cb (C++ type), 334
- ir_learn_state_t (C++ enum), 334
- ir_learn_state_t::IR_LEARN_STATE_END (C++ enumerator), 334
- ir_learn_state_t::IR_LEARN_STATE_EXIT (C++ enumerator), 334
- ir_learn_state_t::IR_LEARN_STATE_FAIL (C++ enumerator), 334
- ir_learn_state_t::IR_LEARN_STATE_READY (C++ enumerator), 334
- ir_learn_state_t::IR_LEARN_STATE_STEP (C++ enumerator), 334
- ir_learn_stop (C++ function), 331
- ir_learn_sub_list_t (C++ struct), 332
- ir_learn_sub_list_t::symbols (C++ member), 333
- ir_learn_sub_list_t::timediff (C++ member), 333
- ## K
- keyboard_btn_callback_t (C++ type), 315
- keyboard_btn_cb_config_t (C++ struct), 314
- keyboard_btn_cb_config_t::callback (C++ member), 314
- keyboard_btn_cb_config_t::event (C++ member), 314
- keyboard_btn_cb_config_t::event_data (C++ member), 314
- keyboard_btn_cb_config_t::user_data (C++ member), 314
- keyboard_btn_cb_handle_t (C++ type), 315
- keyboard_btn_config_t (C++ struct), 314
- keyboard_btn_config_t::active_level (C++ member), 314
- keyboard_btn_config_t::core_id (C++ member), 315
- keyboard_btn_config_t::debounce_ticks (C++ member), 314
- keyboard_btn_config_t::enable_power_save (C++ member), 315
- keyboard_btn_config_t::input_gpio_num (C++ member), 314

keyboard_btn_config_t::input_gpios (C++ member), 314
 keyboard_btn_config_t::output_gpio_num (C++ member), 314
 keyboard_btn_config_t::output_gpios (C++ member), 314
 keyboard_btn_config_t::priority (C++ member), 315
 keyboard_btn_config_t::ticks_interval (C++ member), 314
 keyboard_btn_data_t (C++ struct), 313
 keyboard_btn_data_t::input_index (C++ member), 313
 keyboard_btn_data_t::output_index (C++ member), 313
 keyboard_btn_event_data_t (C++ union), 312
 keyboard_btn_event_data_t::combination (C++ member), 313
 keyboard_btn_event_data_t::combination_key_data (C++ struct), 313
 keyboard_btn_event_data_t::combination_key_data (C++ member), 313
 keyboard_btn_event_data_t::combination_key_num (C++ member), 313
 keyboard_btn_event_t (C++ enum), 315
 keyboard_btn_event_t::KBD_EVENT_COMBINATION (C++ enumerator), 315
 keyboard_btn_event_t::KBD_EVENT_MAX (C++ enumerator), 315
 keyboard_btn_event_t::KBD_EVENT_PRESSED (C++ enumerator), 315
 keyboard_btn_handle_t (C++ type), 315
 keyboard_btn_report_t (C++ struct), 313
 keyboard_btn_report_t::key_change_num (C++ member), 313
 keyboard_btn_report_t::key_data (C++ member), 313
 keyboard_btn_report_t::key_pressed_num (C++ member), 313
 keyboard_btn_report_t::key_release_data (C++ member), 314
 keyboard_btn_report_t::key_release_num (C++ member), 313
 keyboard_button_create (C++ function), 311
 keyboard_button_delete (C++ function), 311
 keyboard_button_get_gpio_by_index (C++ function), 312
 keyboard_button_get_index_by_gpio (C++ function), 312
 keyboard_button_register_cb (C++ function), 311
 keyboard_button_unregister_cb (C++ function), 312
 knob_cb_t (C++ type), 321
 knob_config_t (C++ struct), 321
 knob_config_t::default_direction (C++ member), 321
 knob_config_t::enable_power_save (C++ member), 321
 knob_config_t::gpio_encoder_a (C++ member), 321
 knob_config_t::gpio_encoder_b (C++ member), 321
 knob_event_t (C++ enum), 321
 knob_event_t::KNOB_EVENT_MAX (C++ enumerator), 322
 knob_event_t::KNOB_H_LIM (C++ enumerator), 321
 knob_event_t::KNOB_L_LIM (C++ enumerator), 322
 knob_event_t::KNOB_LEFT (C++ enumerator), 321
 knob_event_t::KNOB_NONE (C++ enumerator), 322
 knob_event_t::KNOB_RIGHT (C++ enumerator), 321
 knob_event_t::KNOB_ZERO (C++ enumerator), 322
 knob_event_t::KNOB_ZERO (C++ enumerator), 322
 knob_event_t (C++ type), 321
 L
 LCD_CMD_LEV (C macro), 21
 LCD_DATA_LEV (C macro), 21
 LED_INDICATOR_CHECK (C macro), 172
 LED_INDICATOR_CHECK_WARNING (C macro), 172
 led_indicator_delete (C++ function), 169
 led_indicator_get_brightness (C++ function), 170
 led_indicator_get_hsv (C++ function), 170
 led_indicator_get_rgb (C++ function), 171
 led_indicator_preempt_start (C++ function), 169
 led_indicator_preempt_stop (C++ function), 170
 led_indicator_set_brightness (C++ function), 170
 led_indicator_set_color_temperature (C++ function), 171
 led_indicator_set_hsv (C++ function), 171
 led_indicator_set_on_off (C++ function), 170
 led_indicator_set_rgb (C++ function), 171
 led_indicator_start (C++ function), 169
 led_indicator_stop (C++ function), 169
 lightbulb_basic_effect_start (C++ function), 444
 lightbulb_basic_effect_stop (C++ function), 444
 lightbulb_basic_effect_stop_and_restore (C++ function), 444
 lightbulb_capability_t (C++ struct), 448
 lightbulb_capability_t::disable_auto_on (C++ member), 448
 lightbulb_capability_t::enable_fade (C++ member), 448

[lightbulb_capability_t::enable_hardware_light](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::enable_lowpower](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::enable_precise_lightbulb](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::enable_precise_lightbulb_color](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::enable_status_strobe](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::fade_time_ms](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::led_beads](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::storage_cb](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::storage_delay_ms](#) ([C++ member](#)), [448](#)
[lightbulb_capability_t::sync_change_brightness](#) ([C++ member](#)), [448](#)
[lightbulb_cct_kelvin_range_t](#) ([C++ struct](#)), [447](#)
[lightbulb_cct_kelvin_range_t::max](#) ([C++ member](#)), [448](#)
[lightbulb_cct_kelvin_range_t::min](#) ([C++ member](#)), [448](#)
[lightbulb_cct_mapping_data_t](#) ([C++ struct](#)), [445](#)
[lightbulb_cct_mapping_data_t::cct_kelvin](#) ([C++ member](#)), [445](#)
[lightbulb_cct_mapping_data_t::cct_percent](#) ([C++ member](#)), [445](#)
[lightbulb_cct_mapping_data_t::rgbcw](#) ([C++ member](#)), [445](#)
[lightbulb_chip_out_pin_t](#) ([C++ enum](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT1](#) ([C++ enumerator](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT2](#) ([C++ enumerator](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT3](#) ([C++ enumerator](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT4](#) ([C++ enumerator](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT5](#) ([C++ enumerator](#)), [455](#)
[lightbulb_chip_out_pin_t::OUT_MAX](#) ([C++ enumerator](#)), [455](#)
[lightbulb_color_mapping_data_t](#) ([C++ struct](#)), [445](#)
[lightbulb_color_mapping_data_t::hue](#) ([C++ member](#)), [445](#)
[lightbulb_color_mapping_data_t::rgbcw_0](#) ([C++ member](#)), [445](#)
[lightbulb_color_mapping_data_t::rgbcw_100](#) ([C++ member](#)), [445](#)
[lightbulb_color_mapping_data_t::rgbcw_500](#) ([C++ member](#)), [445](#)
[lightbulb_config_t](#) ([C++ struct](#)), [449](#)
[lightbulb_config_t::blue](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::capability](#) ([C++ member](#)), [451](#)
[lightbulb_config_t::cct_mix_mode](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::cold_cct](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::cold_white](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::color_mix_mode](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::driver_conf](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::external_limit](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::gamma_conf](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::green](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::iic_io](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::init_status](#) ([C++ member](#)), [451](#)
[lightbulb_config_t::io_conf](#) ([C++ member](#)), [451](#)
[lightbulb_config_t::kelvin_max](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::kelvin_min](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::precise](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::pwm_io](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::red](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::sm16825e_io](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::standard](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::table](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::table_size](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::type](#) ([C++ member](#)), [449](#)
[lightbulb_config_t::warm_brightness](#) ([C++ member](#)), [450](#)
[lightbulb_config_t::warm_yellow](#) ([C++ member](#)), [450](#)
[lightbulb_deinit](#) ([C++ function](#)), [440](#)
[lightbulb_driver_t](#) ([C++ enum](#)), [452](#)
[lightbulb_driver_t::DRIVER_BP1658CJ](#) ([C++ enumerator](#)), [452](#)
[lightbulb_driver_t::DRIVER_BP57x8D](#) ([C++ enumerator](#)), [452](#)
[lightbulb_driver_t::DRIVER_ESP_PWM](#) ([C++ enumerator](#)), [452](#)
[lightbulb_driver_t::DRIVER_KP18058](#) ([C++ enumerator](#)), [452](#)

- (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SELECT_INVALID (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SELECT_MAX (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SM16825E (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SM2135E (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SM2135EH (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SM2182E (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_SM2x35EGH (C++ enumerator), 452
- lightbulb_driver_t::DRIVER_WS2812 (C++ enumerator), 452
- lightbulb_effect_config_t (C++ struct), 451
- lightbulb_effect_config_t::cct (C++ member), 451
- lightbulb_effect_config_t::effect_cycle_ms (C++ member), 451
- lightbulb_effect_config_t::effect_type (C++ member), 451
- lightbulb_effect_config_t::hue (C++ member), 451
- lightbulb_effect_config_t::interrupt_forbidden (C++ member), 451
- lightbulb_effect_config_t::max_value_brightness (C++ member), 451
- lightbulb_effect_config_t::min_value_brightness (C++ member), 451
- lightbulb_effect_config_t::mode (C++ member), 451
- lightbulb_effect_config_t::saturation (C++ member), 451
- lightbulb_effect_config_t::total_ms (C++ member), 451
- lightbulb_effect_config_t::user_cb (C++ member), 451
- lightbulb_effect_t (C++ enum), 453
- lightbulb_effect_t::EFFECT_BLINK (C++ enumerator), 453
- lightbulb_effect_t::EFFECT_BREATH (C++ enumerator), 453
- lightbulb_gamma_config_t (C++ struct), 445
- lightbulb_gamma_config_t::balance_coefficient (C++ member), 446
- lightbulb_gamma_config_t::color_curve_coefficient (C++ member), 446
- lightbulb_gamma_config_t::white_curve_coefficient (C++ member), 446
- lightbulb_get_all_detail (C++ function), 444
- lightbulb_get_brightness (C++ function), 443
- lightbulb_get_cct_kelvin (C++ function), 443
- lightbulb_get_cct_percentage (C++ function), 443
- lightbulb_get_fades_function_status (C++ function), 440
- lightbulb_get_hue (C++ function), 443
- lightbulb_get_mode (C++ function), 444
- lightbulb_get_power_limit (C++ function), 444
- lightbulb_get_saturation (C++ function), 443
- lightbulb_get_switch (C++ function), 443
- lightbulb_get_value (C++ function), 443
- lightbulb_hsv2rgb (C++ function), 440
- lightbulb_iic_out_pin_t (C++ type), 452
- lightbulb_init (C++ function), 440
- lightbulb_kelvin2percentage (C++ function), 441
- lightbulb_led_beads_comb_t (C++ enum), 452
- lightbulb_led_beads_comb_t::LED_BEADS_1CH_C (C++ enumerator), 452
- lightbulb_led_beads_comb_t::LED_BEADS_1CH_W (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_2CH_CW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_3CH_RGB (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_4CH_RGBC (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_4CH_RGBCC (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_4CH_RGBW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_4CH_RGBWW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_5CH_RGBC (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_5CH_RGBCC (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_5CH_RGBCW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_5CH_RGBW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_5CH_RGBWW (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_INVALID (C++ enumerator), 453
- lightbulb_led_beads_comb_t::LED_BEADS_MAX (C++ enumerator), 453
- lightbulb_lighting_output_test (C++ function), 444
- lightbulb_lighting_unit_t (C++ enum), 454
- lightbulb_lighting_unit_t::LIGHTING_ALEXA (C++ enumerator), 454
- lightbulb_lighting_unit_t::LIGHTING_ALL_UNIT (C++ enumerator), 454
- lightbulb_lighting_unit_t::LIGHTING_BASIC_FIVE (C++ enumerator), 454

lightbulb_lighting_unit_t::LIGHTING_COLD_TO_WARM_status_t::brightness (C++
(C++ enumerator), 454 member), 447

lightbulb_lighting_unit_t::LIGHTING_COLOR_EFFECT_status_t::cct_percentage
(C++ enumerator), 454 (C++ member), 447

lightbulb_lighting_unit_t::LIGHTING_COLOR_MUSICAL_WHITE_status_t::hue (C++ member), 447
(C++ enumerator), 454 lightbulb_status_t::mode (C++ member), 446

lightbulb_lighting_unit_t::LIGHTING_COLOR_VALUE_INCREMENT
(C++ enumerator), 454 lightbulb_status_t::on (C++ member), 447

lightbulb_lighting_unit_t::LIGHTING_RAINBOW_status_t::saturation (C++
(C++ enumerator), 454 member), 447

lightbulb_lighting_unit_t::LIGHTING_WARM_TO_COLD_status_t::value (C++ member),
(C++ enumerator), 454 447

lightbulb_lighting_unit_t::LIGHTING_WHITE_BRIGHTNESS_INCREMENT (C++ function), 440
(C++ enumerator), 454 lightbulb_works_mode_t (C++ enum), 454

lightbulb_lighting_unit_t::LIGHTING_WHITE_EFFECT_works_mode_t::WORK_COLOR
(C++ enumerator), 454 (C++ enumerator), 454

lightbulb_percentage2kelvin (C++ function), 442 lightbulb_works_mode_t::WORK_INVALID
(C++ enumerator), 454

lightbulb_power_limit_t (C++ struct), 447 lightbulb_works_mode_t::WORK_WHITE
(C++ enumerator), 454

lightbulb_power_limit_t::color_max_power
(C++ member), 447 lightbulb_xyy2rgb (C++ function), 441

lightbulb_power_limit_t::color_max_value
(C++ member), 447

lightbulb_power_limit_t::color_min_value
(C++ member), 447

lightbulb_power_limit_t::white_max_brightness 414
(C++ member), 447

lightbulb_power_limit_t::white_max_power
(C++ member), 447

lightbulb_power_limit_t::white_min_brightness
(C++ member), 447

lightbulb_rgb2hsv (C++ function), 441

lightbulb_rgb2xyy (C++ function), 441

lightbulb_set_brightness (C++ function), 442

lightbulb_set_cct (C++ function), 442

lightbulb_set_cctb (C++ function), 443

lightbulb_set_fade_time (C++ function), 440

lightbulb_set_fades_function (C++ function), 440

lightbulb_set_hsv (C++ function), 443

lightbulb_set_hue (C++ function), 442

lightbulb_set_saturation (C++ function), 442

lightbulb_set_storage_function (C++ function), 440

lightbulb_set_switch (C++ function), 443

lightbulb_set_value (C++ function), 442

lightbulb_set_xyy (C++ function), 442

lightbulb_status_erase_nvs_storage
(C++ function), 444

lightbulb_status_get_from_nvs (C++ function), 444

lightbulb_status_set_to_nvs (C++ function), 444

lightbulb_status_storage_cb_t (C++ type), 452

lightbulb_status_t (C++ struct), 446

MAX_BLE_DEVNAME_LEN (C macro), 35

MAX_SPEED_MEASUREMENT_FACTOR (C macro), 414

MCPWM_CLK_SRC (C macro), 411

MCPWM_PERIOD (C macro), 411

mic_callback_t (C++ type), 223

mic_frame_t (C++ struct), 219

mic_frame_t::bit_resolution (C++ member), 219

mic_frame_t::data (C++ member), 219

mic_frame_t::data_bytes (C++ member), 219

mic_frame_t::samples_frequence (C++ member), 219

MIN (C macro), 36

mmap_assets_config_t (C++ struct), 180

mmap_assets_config_t::app_bin_check
(C++ member), 180

mmap_assets_config_t::checksum (C++ member), 180

mmap_assets_config_t::flags (C++ member), 181

mmap_assets_config_t::full_check (C++ member), 180

mmap_assets_config_t::max_files (C++ member), 180

mmap_assets_config_t::metadata_check
(C++ member), 181

mmap_assets_config_t::mmap_enable
(C++ member), 180

mmap_assets_config_t::partition_label
(C++ member), 180

mmap_assets_config_t::reserved (C++ member), 181

mmap_assets_copy_mem (C++ function), 179

mmap_assets_del (C++ function), 179

[mmap_assets_get_height \(C++ function\), 180](#)
[mmap_assets_get_mem \(C++ function\), 179](#)
[mmap_assets_get_name \(C++ function\), 179](#)
[mmap_assets_get_size \(C++ function\), 179](#)
[mmap_assets_get_stored_files \(C++ function\), 180](#)
[mmap_assets_get_width \(C++ function\), 180](#)
[mmap_assets_handle_t \(C++ type\), 181](#)
[mmap_assets_new \(C++ function\), 179](#)

N

[ntc_circuit_mode_t \(C++ enum\), 357](#)
[ntc_circuit_mode_t::CIRCUIT_MODE_NTC_GND \(C++ enumerator\), 357](#)
[ntc_circuit_mode_t::CIRCUIT_MODE_NTC_VCC \(C++ enumerator\), 357](#)
[ntc_config_t \(C++ struct\), 356](#)
[ntc_config_t::atten \(C++ member\), 356](#)
[ntc_config_t::b_value \(C++ member\), 357](#)
[ntc_config_t::channel \(C++ member\), 356](#)
[ntc_config_t::circuit_mode \(C++ member\), 356](#)
[ntc_config_t::fixed_ohm \(C++ member\), 357](#)
[ntc_config_t::r25_ohm \(C++ member\), 357](#)
[ntc_config_t::unit \(C++ member\), 356](#)
[ntc_config_t::vdd_mv \(C++ member\), 357](#)
[ntc_dev_create \(C++ function\), 355](#)
[ntc_dev_delete \(C++ function\), 356](#)
[ntc_dev_get_adc_handle \(C++ function\), 356](#)
[ntc_dev_get_temperature \(C++ function\), 356](#)
[ntc_device_handle_t \(C++ type\), 357](#)
[NULL_I2C_DEV_ADDR \(C macro\), 13](#)
[NULL_I2C_MEM_16BIT_ADDR \(C macro\), 13](#)
[NULL_I2C_MEM_ADDR \(C macro\), 13](#)
[NULL_SPI_CS_PIN \(C macro\), 18](#)
[nvs_modified_cb_t \(C++ type\), 253](#)

O

[OpenAI \(C++ struct\), 287](#)
[OpenAI_Audio_Input_Format \(C++ enum\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_M4A \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_MP3 \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_MP4 \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_MPEG \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_MPGA \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_WAV \(C++ enumerator\), 289](#)
[OpenAI_Audio_Input_Format::OPENAI_AUDIO_INPUT_FORMAT_WEBM \(C++ enumerator\), 289](#)
[OpenAI_Audio_Output_Format \(C++ enum\), 289](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_M4A \(C++ enumerator\), 290](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_MP3 \(C++ enumerator\), 290](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_MP4 \(C++ enumerator\), 289](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_MPEG \(C++ enumerator\), 290](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_MPGA \(C++ enumerator\), 290](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_WAV \(C++ enumerator\), 290](#)
[OpenAI_Audio_Output_Format::OPENAI_AUDIO_OUTPUT_FORMAT_WEBM \(C++ enumerator\), 290](#)
[OpenAI_Audio_Response_Format \(C++ enum\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_M4A \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_MP3 \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_MP4 \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_MPEG \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_MPGA \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_WAV \(C++ enumerator\), 289](#)
[OpenAI_Audio_Response_Format::OPENAI_AUDIO_RESPONSE_FORMAT_WEBM \(C++ enumerator\), 289](#)
[OpenAI_AudioSpeech \(C++ struct\), 286](#)
[OpenAI_AudioSpeech::setModel \(C++ member\), 286](#)
[OpenAI_AudioSpeech::setResponseFormat \(C++ member\), 286](#)
[OpenAI_AudioSpeech::setSpeed \(C++ member\), 286](#)
[OpenAI_AudioSpeech::setVoice \(C++ member\), 286](#)
[OpenAI_AudioSpeech::speech \(C++ member\), 286](#)
[OpenAI_AudioSpeech::speechStream \(C++ member\), 286](#)
[OpenAI_AudioSpeech_t \(C++ type\), 288](#)
[OpenAI_AudioTranscription \(C++ struct\), 285](#)
[OpenAI_AudioTranscription::file \(C++ member\), 286](#)
[OpenAI_AudioTranscription::setLanguage \(C++ member\), 285](#)
[OpenAI_AudioTranscription::setPrompt \(C++ member\), 285](#)
[OpenAI_AudioTranscription::setResponseFormat \(C++ member\), 285](#)
[OpenAI_AudioTranscription::setTemperature \(C++ member\), 285](#)
[OpenAI_AudioTranscription_t \(C++ type\), 288](#)
[OpenAI_AudioTranslation \(C++ struct\), 287](#)
[OpenAI_AudioTranslation::file \(C++ member\), 287](#)
[OpenAI_AudioTranslation::setPrompt \(C++ member\), 287](#)
[OpenAI_AudioTranslation::setResponseFormat \(C++ member\), 287](#)

OpenAI_AudioTranslation::setTemperature (C++ member), 287
 OpenAI_AudioTranslation_t (C++ type), 288
 OpenAI_ChatCompletion (C++ struct), 281
 OpenAI_ChatCompletion::clearConversation (C++ member), 282
 OpenAI_ChatCompletion::message (C++ member), 282
 OpenAI_ChatCompletion::multiModalMessage (C++ member), 282
 OpenAI_ChatCompletion::setFrequencyPenalty (C++ member), 281
 OpenAI_ChatCompletion::setMaxTokens (C++ member), 281
 OpenAI_ChatCompletion::setModel (C++ member), 281
 OpenAI_ChatCompletion::setPresencePenalty (C++ member), 281
 OpenAI_ChatCompletion::setStop (C++ member), 281
 OpenAI_ChatCompletion::setSystem (C++ member), 281
 OpenAI_ChatCompletion::setTemperature (C++ member), 281
 OpenAI_ChatCompletion::setTopP (C++ member), 281
 OpenAI_ChatCompletion::setUser (C++ member), 282
 OpenAI_ChatCompletion_t (C++ type), 288
 OpenAI_Completion (C++ struct), 279
 OpenAI_Completion::prompt (C++ member), 280
 OpenAI_Completion::setBestOf (C++ member), 280
 OpenAI_Completion::setEcho (C++ member), 280
 OpenAI_Completion::setFrequencyPenalty (C++ member), 280
 OpenAI_Completion::setMaxTokens (C++ member), 279
 OpenAI_Completion::setModel (C++ member), 279
 OpenAI_Completion::setN (C++ member), 280
 OpenAI_Completion::setPresencePenalty (C++ member), 280
 OpenAI_Completion::setStop (C++ member), 280
 OpenAI_Completion::setTemperature (C++ member), 279
 OpenAI_Completion::setTopP (C++ member), 280
 OpenAI_Completion::setUser (C++ member), 280
 OpenAI_Completion_t (C++ type), 288
 OpenAI_Edit (C++ struct), 282
 OpenAI_Edit::process (C++ member), 283
 OpenAI_Edit::setModel (C++ member), 282
 OpenAI_Edit::setN (C++ member), 283
 OpenAI_Edit::setTemperature (C++ member), 282
 OpenAI_Edit::setTopP (C++ member), 282
 OpenAI_Edit_t (C++ type), 288
 OpenAI_EmbeddingData_t (C++ struct), 276
 OpenAI_EmbeddingData_t::data (C++ member), 276
 OpenAI_EmbeddingData_t::len (C++ member), 276
 OpenAI_EmbeddingResponse (C++ struct), 276
 OpenAI_EmbeddingResponse::deleteResponse (C++ member), 277
 OpenAI_EmbeddingResponse::getData (C++ member), 277
 OpenAI_EmbeddingResponse::getError (C++ member), 277
 OpenAI_EmbeddingResponse::getLen (C++ member), 276
 OpenAI_EmbeddingResponse::getUsage (C++ member), 276
 OpenAI_EmbeddingResponse_t (C++ type), 287
 OpenAI_Image_Response_Format (C++ enum), 289
 OpenAI_Image_Response_Format::OPENAI_IMAGE_RESPONSE_FORMAT_1 (C++ enumerator), 289
 OpenAI_Image_Response_Format::OPENAI_IMAGE_RESPONSE_FORMAT_2 (C++ enumerator), 289
 OpenAI_Image_Size (C++ enum), 288
 OpenAI_Image_Size::OPENAI_IMAGE_SIZE_1024x1024 (C++ enumerator), 288
 OpenAI_Image_Size::OPENAI_IMAGE_SIZE_256x256 (C++ enumerator), 288
 OpenAI_Image_Size::OPENAI_IMAGE_SIZE_512x512 (C++ enumerator), 288
 OpenAI_ImageEdit (C++ struct), 284
 OpenAI_ImageEdit::image (C++ member), 285
 OpenAI_ImageEdit::setN (C++ member), 285
 OpenAI_ImageEdit::setPrompt (C++ member), 284
 OpenAI_ImageEdit::setResponseFormat (C++ member), 285
 OpenAI_ImageEdit::setSize (C++ member), 284
 OpenAI_ImageEdit::setUser (C++ member), 285
 OpenAI_ImageEdit_t (C++ type), 288
 OpenAI_ImageGeneration (C++ struct), 283
 OpenAI_ImageGeneration::prompt (C++ member), 283
 OpenAI_ImageGeneration::setN (C++ member), 283
 OpenAI_ImageGeneration::setResponseFormat (C++ member), 283
 OpenAI_ImageGeneration::setSize (C++ member), 283
 OpenAI_ImageGeneration::setUser (C++ member), 283
 OpenAI_ImageGeneration_t (C++ type), 288

- OpenAI_ImageResponse (C++ struct), 277
 OpenAI_ImageResponse::deleteResponse (C++ member), 278
 OpenAI_ImageResponse::getData (C++ member), 278
 OpenAI_ImageResponse::getError (C++ member), 278
 OpenAI_ImageResponse::getLen (C++ member), 278
 OpenAI_ImageResponse_t (C++ type), 288
 OpenAI_ImageVariation (C++ struct), 283
 OpenAI_ImageVariation::image (C++ member), 284
 OpenAI_ImageVariation::setN (C++ member), 284
 OpenAI_ImageVariation::setResponseFormat (C++ member), 284
 OpenAI_ImageVariation::setSize (C++ member), 284
 OpenAI_ImageVariation::setUser (C++ member), 284
 OpenAI_ImageVariation_t (C++ type), 288
 OpenAI_ModerationResponse (C++ struct), 277
 OpenAI_ModerationResponse::deleteResponse (C++ member), 277
 OpenAI_ModerationResponse::getData (C++ member), 277
 OpenAI_ModerationResponse::getError (C++ member), 277
 OpenAI_ModerationResponse::getLen (C++ member), 277
 OpenAI_ModerationResponse_t (C++ type), 287
 OpenAI_SpeechResponse (C++ struct), 279
 OpenAI_SpeechResponse::deleteResponse (C++ member), 279
 OpenAI_SpeechResponse::getData (C++ member), 279
 OpenAI_SpeechResponse::getLen (C++ member), 279
 OpenAI_SpeechResponse_t (C++ type), 288
 OpenAI_StreamCallback (C++ type), 287
 OpenAI_StringResponse (C++ struct), 278
 OpenAI_StringResponse::deleteResponse (C++ member), 279
 OpenAI_StringResponse::getData (C++ member), 278
 OpenAI_StringResponse::getError (C++ member), 278
 OpenAI_StringResponse::getLen (C++ member), 278
 OpenAI_StringResponse::getUsage (C++ member), 278
 OpenAI_StringResponse_t (C++ type), 288
 OpenAI_t (C++ type), 288
 OpenAIChangeBaseURL (C++ function), 276
 OpenAICreate (C++ function), 276
 OpenAIDelete (C++ function), 276
 overcurrent_event_data_t (C++ struct), 362
 overcurrent_event_data_t::current_value (C++ member), 362
 overcurrent_event_data_t::trigger_value (C++ member), 362
 overvoltage_event_data_t (C++ struct), 362
 overvoltage_event_data_t::overvalue (C++ member), 362
 overvoltage_event_data_t::voltage_value (C++ member), 362
- ## P
- POLE_PAIR (C macro), 413
 portTICK_RATE_MS (C macro), 14
 power_measure_calibration_factor_reset (C++ function), 361
 power_measure_chip_t (C++ enum), 365
 power_measure_chip_t::CHIP_BL0937 (C++ enumerator), 365
 power_measure_chip_t::CHIP_MAX (C++ enumerator), 365
 power_measure_deinit (C++ function), 360
 power_measure_event_t (C++ enum), 364
 power_measure_event_t::POWER_MEASURE_ENERGY_REPORT (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_HOME_APPLIANC (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_HOME_APPLIANC (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_INIT_DONE (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_OVERCURRENT (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_OVERVOLTAGE (C++ enumerator), 364
 power_measure_event_t::POWER_MEASURE_UNDERVOLTAGE (C++ enumerator), 364
 power_measure_get_active_power (C++ function), 360
 power_measure_get_calibration_factor (C++ function), 361
 power_measure_get_current (C++ function), 360
 power_measure_get_energy (C++ function), 361
 power_measure_get_power_factor (C++ function), 360
 power_measure_get_voltage (C++ function), 360
 power_measure_init (C++ function), 360
 power_measure_init_config_t (C++ struct), 363
 power_measure_init_config_t::chip_config (C++ member), 364
 power_measure_init_config_t::enable_energy_detect (C++ member), 364
 power_measure_init_config_t::overcurrent (C++ member), 364

- power_measure_init_config_t::overvoltage (C++ member), 364
- power_measure_init_config_t::undervoltage (C++ member), 364
- power_measure_rated_voltage_t (C++ enum), 365
- power_measure_rated_voltage_t::RATED_VOLTAGE_100V (C++ enumerator), 365
- power_measure_rated_voltage_t::RATED_VOLTAGE_120V (C++ enumerator), 365
- power_measure_rated_voltage_t::RATED_VOLTAGE_220V (C++ enumerator), 365
- power_measure_reset_energy_calculation (C++ function), 361
- power_measure_start_calibration (C++ function), 361
- power_measure_start_energy_calculation (C++ function), 361
- power_measure_stop_energy_calculation (C++ function), 361
- proxi_cb_t (C++ type), 375
- proxi_state_t (C++ enum), 376
- proxi_state_t::PROXI_STATE_ACTIVE (C++ enumerator), 376
- proxi_state_t::PROXI_STATE_INACTIVE (C++ enumerator), 376
- pwm_audio_channel_t (C++ enum), 259
- pwm_audio_channel_t::PWM_AUDIO_CH_MAX (C++ enumerator), 259
- pwm_audio_channel_t::PWM_AUDIO_CH_MONO (C++ enumerator), 259
- pwm_audio_channel_t::PWM_AUDIO_CH_STEREO (C++ enumerator), 259
- pwm_audio_config_t (C++ struct), 258
- pwm_audio_config_t::duty_resolution (C++ member), 259
- pwm_audio_config_t::gpio_num_left (C++ member), 258
- pwm_audio_config_t::gpio_num_right (C++ member), 258
- pwm_audio_config_t::ledc_channel_left (C++ member), 259
- pwm_audio_config_t::ledc_channel_right (C++ member), 259
- pwm_audio_config_t::ledc_timer_sel (C++ member), 259
- pwm_audio_config_t::ringbuf_len (C++ member), 259
- pwm_audio_deinit (C++ function), 257
- pwm_audio_get_param (C++ function), 258
- pwm_audio_get_status (C++ function), 258
- pwm_audio_get_volume (C++ function), 258
- pwm_audio_init (C++ function), 256
- pwm_audio_set_param (C++ function), 257
- pwm_audio_set_sample_rate (C++ function), 258
- pwm_audio_set_volume (C++ function), 258
- pwm_audio_start (C++ function), 257
- pwm_audio_status_t (C++ enum), 259
- pwm_audio_status_t::PWM_AUDIO_STATUS_BUSY (C++ enumerator), 259
- pwm_audio_status_t::PWM_AUDIO_STATUS_IDLE (C++ enumerator), 259
- pwm_audio_status_t::PWM_AUDIO_STATUS_UN_INIT (C++ enumerator), 259
- pwm_audio_stop (C++ function), 257
- pwm_audio_write (C++ function), 257
- PWM_DUTYCYCLE_05 (C macro), 411
- PWM_DUTYCYCLE_10 (C macro), 411
- PWM_DUTYCYCLE_100 (C macro), 412
- PWM_DUTYCYCLE_15 (C macro), 411
- PWM_DUTYCYCLE_20 (C macro), 411
- PWM_DUTYCYCLE_25 (C macro), 411
- PWM_DUTYCYCLE_30 (C macro), 411
- PWM_DUTYCYCLE_40 (C macro), 412
- PWM_DUTYCYCLE_50 (C macro), 412
- PWM_DUTYCYCLE_60 (C macro), 412
- PWM_DUTYCYCLE_80 (C macro), 412
- PWM_DUTYCYCLE_90 (C macro), 412
- PWM_MODE (C macro), 411
- ## R
- RAMP_DUTY_END (C macro), 413
- RAMP_DUTY_INC (C macro), 413
- RAMP_DUTY_STA (C macro), 412
- RAMP_TIM_END (C macro), 412
- RAMP_TIM_STA (C macro), 412
- RAMP_TIM_STEP (C macro), 412
- ## S
- sensor_command_t (C++ enum), 346
- sensor_command_t::COMMAND_MAX (C++ enumerator), 346
- sensor_command_t::COMMAND_SELF_TEST (C++ enumerator), 346
- sensor_command_t::COMMAND_SET_MODE (C++ enumerator), 346
- sensor_command_t::COMMAND_SET_ODR (C++ enumerator), 346
- sensor_command_t::COMMAND_SET_POWER (C++ enumerator), 346
- sensor_command_t::COMMAND_SET_RANGE (C++ enumerator), 346
- sensor_config_t (C++ struct), 351
- sensor_config_t::addr (C++ member), 351
- sensor_config_t::bus (C++ member), 351
- sensor_config_t::intr_pin (C++ member), 351
- sensor_config_t::intr_type (C++ member), 351
- sensor_config_t::min_delay (C++ member), 351
- sensor_config_t::mode (C++ member), 351
- sensor_config_t::range (C++ member), 351
- sensor_config_t::type (C++ member), 351
- sensor_data_event_id_t (C++ enum), 347

sensor_data_event_id_t::SENSOR_ACCE_DATA_READY (C++ enumerator), 347
 sensor_data_event_id_t::SENSOR_BARO_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_CURRENT_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_EVENT_ID_END (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_FORCE_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_GYRO_DATA_READY (C++ enumerator), 347
 sensor_data_event_id_t::SENSOR_HR_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_HUMI_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_LIGHT_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_MAG_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_NOISE_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_PROXI_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_RGBW_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_STEP_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_TEMP_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_TVOC_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_UV_DATA_READY (C++ enumerator), 348
 sensor_data_event_id_t::SENSOR_VOLTAGE_DATA_READY (C++ enumerator), 348
 sensor_data_group_t (C++ struct), 345
 sensor_data_group_t::number (C++ member), 345
 sensor_data_group_t::sensor_data (C++ member), 345
 sensor_data_t (C++ struct), 343
 sensor_data_t::acce (C++ member), 343
 sensor_data_t::baro (C++ member), 344
 sensor_data_t::current (C++ member), 344
 sensor_data_t::data (C++ member), 345
 sensor_data_t::event_id (C++ member), 343
 sensor_data_t::force (C++ member), 344
 sensor_data_t::gyro (C++ member), 344
 sensor_data_t::hr (C++ member), 344
 sensor_data_t::humidity (C++ member), 344
 sensor_data_t::light (C++ member), 344
 sensor_data_t::mag (C++ member), 344
 sensor_data_t::min_delay (C++ member), 343
 sensor_data_t::noise (C++ member), 344
 sensor_data_t::proximity (C++ member), 344
 sensor_data_t::rgbw (C++ member), 344
 sensor_data_t::sensor_addr (C++ member), 343
 sensor_data_t::sensor_name (C++ member), 343
 sensor_data_t::sensor_type (C++ member), 344
 sensor_data_t::step (C++ member), 344
 sensor_data_t::temperature (C++ member), 344
 sensor_data_t::timestamp (C++ member), 343
 sensor_data_t::tvoc (C++ member), 344
 sensor_data_t::uv (C++ member), 344
 sensor_data_t::voltage (C++ member), 344
 sensor_device_impl_t (C++ type), 345
 sensor_driver_handle_t (C++ type), 345
 SENSOR_EVENT_ANY_ID (C macro), 345
 sensor_event_handler_instance_t (C++ type), 352
 sensor_event_handler_t (C++ type), 352
 sensor_event_id_t (C++ enum), 347
 sensor_event_id_t::SENSOR_EVENT_COMMON_END (C++ enumerator), 347
 sensor_event_id_t::SENSOR_STARTED (C++ enumerator), 347
 sensor_event_id_t::SENSOR_STOPED (C++ enumerator), 347
 sensor_handle_t (C++ type), 352
 SENSOR_HUB_DETECT_FN (C macro), 352
 sensor_hub_detect_fn_t (C++ struct), 351
 sensor_hub_detect_fn_t::fn (C++ member), 351
 sensor_info_t (C++ struct), 350
 sensor_info_t::name (C++ member), 350
 sensor_info_t::sensor_type (C++ member), 351
 sensor_mode_t (C++ enum), 346
 sensor_mode_t::MODE_DEFAULT (C++ enumerator), 346
 sensor_mode_t::MODE_INTERRUPT (C++ enumerator), 347
 sensor_mode_t::MODE_MAX (C++ enumerator), 347
 sensor_mode_t::MODE_POLLING (C++ enumerator), 347
 sensor_power_mode_t (C++ enum), 346
 sensor_power_mode_t::POWER_MAX (C++ enumerator), 346
 sensor_power_mode_t::POWER_MODE_SLEEP (C++ enumerator), 346
 sensor_power_mode_t::POWER_MODE_WAKEUP (C++ enumerator), 346
 sensor_range_t (C++ enum), 347
 sensor_range_t::RANGE_DEFAULT (C++ enumerator), 347
 sensor_range_t::RANGE_MAX (C++ enumerator), 347

- sensor_range_t::RANGE_MEDIUM (C++ *enumerator*), 347
 - sensor_range_t::RANGE_MIN (C++ *enumerator*), 347
 - sensor_type_t (C++ *enum*), 345
 - sensor_type_t::HUMITURE_ID (C++ *enumerator*), 345
 - sensor_type_t::IMU_ID (C++ *enumerator*), 345
 - sensor_type_t::LIGHT_SENSOR_ID (C++ *enumerator*), 346
 - sensor_type_t::NULL_ID (C++ *enumerator*), 345
 - sensor_type_t::SENSOR_TYPE_MAX (C++ *enumerator*), 346
 - servo_channel_t (C++ *struct*), 416
 - servo_channel_t::ch (C++ *member*), 416
 - servo_channel_t::servo_pin (C++ *member*), 416
 - servo_config_t (C++ *struct*), 417
 - servo_config_t::channel_number (C++ *member*), 417
 - servo_config_t::channels (C++ *member*), 417
 - servo_config_t::freq (C++ *member*), 417
 - servo_config_t::max_angle (C++ *member*), 417
 - servo_config_t::max_width_us (C++ *member*), 417
 - servo_config_t::min_width_us (C++ *member*), 417
 - servo_config_t::timer_number (C++ *member*), 417
 - SPEED_CAL_TYPE (C *macro*), 414
 - SPEED_KD (C *macro*), 413
 - SPEED_KI (C *macro*), 413
 - SPEED_KP (C *macro*), 413
 - SPEED_MAX_INTEGRAL (C *macro*), 414
 - SPEED_MAX_OUTPUT (C *macro*), 414
 - SPEED_MAX_RPM (C *macro*), 414
 - SPEED_MIN_INTEGRAL (C *macro*), 414
 - SPEED_MIN_OUTPUT (C *macro*), 414
 - SPEED_MIN_RPM (C *macro*), 414
 - speed_mode_t (C++ *enum*), 410
 - speed_mode_t::SPEED_CLOSED_LOOP (C++ *enumerator*), 410
 - speed_mode_t::SPEED_OPEN_LOOP (C++ *enumerator*), 410
 - 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*), 16
 - spi_bus_device_handle_t (C++ *type*), 18
 - spi_bus_handle_t (C++ *type*), 18
 - spi_bus_transfer_byte (C++ *function*), 16
 - spi_bus_transfer_bytes (C++ *function*), 16
 - spi_bus_transfer_reg16 (C++ *function*), 16
 - spi_bus_transfer_reg32 (C++ *function*), 17
 - 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*), 18
 - spi_device_config_t::cs_io_num (C++ *member*), 17
 - spi_device_config_t::mode (C++ *member*), 18
 - state_callback_t (C++ *type*), 223
 - static_input_cb_t (C++ *type*), 459
 - STORGE_FACTOR_KEY (C *macro*), 364
 - stream_ctrl_t (C++ *enum*), 224
 - stream_ctrl_t::CTRL_MAX (C++ *enumerator*), 224
 - stream_ctrl_t::CTRL_NONE (C++ *enumerator*), 224
 - stream_ctrl_t::CTRL_RESUME (C++ *enumerator*), 224
 - stream_ctrl_t::CTRL_SUSPEND (C++ *enumerator*), 224
 - stream_ctrl_t::CTRL_UAC_MUTE (C++ *enumerator*), 224
 - stream_ctrl_t::CTRL_UAC_VOLUME (C++ *enumerator*), 224
- ## T
- tinyuf2_nvs_config_t (C++ *struct*), 253
 - tinyuf2_nvs_config_t::modified_cb (C++ *member*), 253
 - tinyuf2_nvs_config_t::namespace_name (C++ *member*), 253
 - tinyuf2_nvs_config_t::part_name (C++ *member*), 253
 - tinyuf2_ota_config_t (C++ *struct*), 252
 - tinyuf2_ota_config_t::complete_cb (C++ *member*), 253
 - tinyuf2_ota_config_t::if_restart (C++ *member*), 253
 - tinyuf2_ota_config_t::label (C++ *member*), 252
 - tinyuf2_ota_config_t::subtype (C++ *member*), 252
 - tinyuf2_state_t (C++ *enum*), 253
 - tinyuf2_state_t::TINYUF2_STATE_INSTALLED (C++ *enumerator*), 253
 - tinyuf2_state_t::TINYUF2_STATE_MOUNTED (C++ *enumerator*), 254
 - tinyuf2_state_t::TINYUF2_STATE_NOT_INSTALLED (C++ *enumerator*), 253
 - touch_button_config_t (C++ *struct*), 379

touch_button_config_t::channel_gold_value (C++ enumerator), 328
 (C++ member), 380
 touch_button_config_t::channel_list (C++ member), 380
 touch_button_config_t::channel_num (C++ member), 380
 touch_button_config_t::channel_threshold (C++ member), 380
 touch_button_config_t::debounce_times (C++ member), 380
 touch_button_config_t::skip_lowlevel_interrupt (C++ member), 380
 touch_button_handle_t (C++ type), 380
 touch_button_sensor_create (C++ function), 378
 touch_button_sensor_delete (C++ function), 378
 touch_button_sensor_get_data (C++ function), 378
 touch_button_sensor_get_state (C++ function), 379
 touch_button_sensor_get_state_bitmap (C++ function), 379
 touch_button_sensor_handle_events (C++ function), 379
 touch_cb_t (C++ type), 380
 TOUCH_MAX_POINT_NUMBER (C macro), 326
 touch_panel_config_t (C++ struct), 324
 touch_panel_config_t::clk_freq (C++ member), 325
 touch_panel_config_t::direction (C++ member), 325
 touch_panel_config_t::height (C++ member), 325
 touch_panel_config_t::i2c_addr (C++ member), 325
 touch_panel_config_t::i2c_bus (C++ member), 325
 touch_panel_config_t::interface_i2c (C++ member), 325
 touch_panel_config_t::interface_spi (C++ member), 325
 touch_panel_config_t::interface_type (C++ member), 325
 touch_panel_config_t::pin_num_cs (C++ member), 325
 touch_panel_config_t::pin_num_int (C++ member), 325
 touch_panel_config_t::spi_bus (C++ member), 325
 touch_panel_config_t::width (C++ member), 325
 touch_panel_config_t::[anonymous] (C++ member), 325
 touch_panel_controller_t (C++ enum), 328
 touch_panel_controller_t::TOUCH_PANEL_CONTROLLER_BT (C++ enumerator), 328
 touch_panel_controller_t::TOUCH_PANEL_CONTROLLER_FT5X06 (C++ enumerator), 328
 touch_panel_controller_t::TOUCH_PANEL_CONTROLLER_I2C (C++ enumerator), 328
 touch_panel_controller_t::TOUCH_PANEL_CONTROLLER_I2C1016 (C++ enumerator), 328
 touch_panel_dir_t (C++ enum), 327
 touch_panel_dir_t::TOUCH_DIR_BTLR (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_BTRL (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_LRBT (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_LRTB (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_MAX (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_RLBT (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_RLTB (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_TBLR (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_DIR_TBRL (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_MIRROR_X (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_MIRROR_Y (C++ enumerator), 327
 touch_panel_dir_t::TOUCH_SWAP_XY (C++ enumerator), 327
 touch_panel_driver_t (C++ struct), 325
 touch_panel_driver_t::calibration_run (C++ member), 326
 touch_panel_driver_t::deinit (C++ member), 326
 touch_panel_driver_t::init (C++ member), 325
 touch_panel_driver_t::read_point_data (C++ member), 326
 touch_panel_driver_t::set_direction (C++ member), 326
 touch_panel_event_t (C++ enum), 326
 touch_panel_event_t::TOUCH_EVT_PRESS (C++ enumerator), 327
 touch_panel_event_t::TOUCH_EVT_RELEASE (C++ enumerator), 327
 touch_panel_find_driver (C++ function), 324
 touch_panel_interface_type_t (C++ enum), 327
 touch_panel_interface_type_t::TOUCH_PANEL_IFACE_I2C (C++ enumerator), 328
 touch_panel_interface_type_t::TOUCH_PANEL_IFACE_SPI (C++ enumerator), 328
 touch_panel_points_t (C++ struct), 324
 touch_panel_points_t::curx (C++ member), 324
 touch_panel_points_t::cury (C++ member), 324
 touch_panel_points_t::event (C++ member), 324
 touch_panel_points_t::event_data (C++ member), 324

touch_panel_points_t::point_num (C++ member), 324
 touch_proxi_config_t (C++ struct), 375
 touch_proxi_config_t::channel_gold_value (C++ member), 375
 touch_proxi_config_t::channel_list (C++ member), 375
 touch_proxi_config_t::channel_num (C++ member), 375
 touch_proxi_config_t::channel_threshold (C++ member), 375
 touch_proxi_config_t::debounce_times (C++ member), 375
 touch_proxi_config_t::skip_lowlevel_init (C++ member), 375
 touch_proximity_handle_t (C++ type), 375
 touch_proximity_sensor_create (C++ function), 374
 touch_proximity_sensor_delete (C++ function), 374
 touch_proximity_sensor_get_data (C++ function), 374
 touch_proximity_sensor_get_state (C++ function), 374
 touch_proximity_sensor_handle_events (C++ function), 375
 touch_slider_config_t (C++ struct), 385
 touch_slider_config_t::calculate_window (C++ member), 386
 touch_slider_config_t::channel_gold_value (C++ member), 385
 touch_slider_config_t::channel_list (C++ member), 385
 touch_slider_config_t::channel_num (C++ member), 385
 touch_slider_config_t::channel_threshold (C++ member), 385
 touch_slider_config_t::debounce_times (C++ member), 386
 touch_slider_config_t::filter_reset_times (C++ member), 386
 touch_slider_config_t::position_range (C++ member), 386
 touch_slider_config_t::skip_lowlevel_init (C++ member), 386
 touch_slider_config_t::swipe_alpha (C++ member), 386
 touch_slider_config_t::swipe_hysteresis (C++ member), 386
 touch_slider_config_t::swipe_threshold (C++ member), 386
 touch_slider_event_cb_t (C++ type), 386
 touch_slider_event_t (C++ enum), 386
 touch_slider_event_t::TOUCH_SLIDER_EVENT_LEFT_SWIPE (C++ enumerator), 387
 touch_slider_event_t::TOUCH_SLIDER_EVENT_NONE (C++ enumerator), 386
 touch_slider_event_t::TOUCH_SLIDER_EVENT_POSITION (C++ enumerator), 387
 touch_slider_event_t::TOUCH_SLIDER_EVENT_RELEASE (C++ enumerator), 387
 touch_slider_event_t::TOUCH_SLIDER_EVENT_RIGHT_SWIPE (C++ enumerator), 387
 touch_slider_handle_t (C++ type), 386
 touch_slider_sensor_create (C++ function), 384
 touch_slider_sensor_delete (C++ function), 384
 touch_slider_sensor_get_data (C++ function), 384
 touch_slider_sensor_get_state (C++ function), 385
 touch_slider_sensor_handle_events (C++ function), 385
 touch_state_t (C++ enum), 380
 touch_state_t::TOUCH_STATE_ACTIVE (C++ enumerator), 380
 touch_state_t::TOUCH_STATE_INACTIVE (C++ enumerator), 380

U

UAC_BITS_ANY (C macro), 222
 UAC_CH_ANY (C macro), 222
 uac_config_t (C++ struct), 220
 uac_config_t::ac_interface (C++ member), 221
 uac_config_t::flags (C++ member), 222
 uac_config_t::mic_bit_resolution (C++ member), 221
 uac_config_t::mic_buf_size (C++ member), 221
 uac_config_t::mic_cb (C++ member), 221
 uac_config_t::mic_cb_arg (C++ member), 221
 uac_config_t::mic_ch_num (C++ member), 220
 uac_config_t::mic_ep_addr (C++ member), 221
 uac_config_t::mic_ep_mps (C++ member), 221
 uac_config_t::mic_fu_id (C++ member), 221
 uac_config_t::mic_interface (C++ member), 221
 uac_config_t::mic_samples_frequence (C++ member), 221
 uac_config_t::spk_bit_resolution (C++ member), 221
 uac_config_t::spk_buf_size (C++ member), 221
 uac_config_t::spk_ch_num (C++ member), 220
 uac_config_t::spk_ep_addr (C++ member), 221
 uac_config_t::spk_ep_mps (C++ member), 221
 uac_config_t::spk_fu_id (C++ member), 222

- `uac_config_t::spk_interface` (C++ member), 221
- `uac_config_t::spk_samples_frequence` (C++ member), 221
- `uac_device_config_t` (C++ struct), 248
- `uac_device_config_t::cb_ctx` (C++ member), 248
- `uac_device_config_t::input_cb` (C++ member), 248
- `uac_device_config_t::output_cb` (C++ member), 248
- `uac_device_config_t::set_mute_cb` (C++ member), 248
- `uac_device_config_t::set_volume_cb` (C++ member), 248
- `uac_device_config_t::skip_tinyusb_init` (C++ member), 248
- `uac_device_init` (C++ function), 248
- `uac_frame_size_list_get` (C++ function), 217
- `uac_frame_size_reset` (C++ function), 217
- `uac_frame_size_t` (C++ struct), 220
- `uac_frame_size_t::bit_resolution` (C++ member), 220
- `uac_frame_size_t::ch_num` (C++ member), 220
- `uac_frame_size_t::samples_frequence` (C++ member), 220
- `uac_frame_size_t::samples_frequence_max` (C++ member), 220
- `uac_frame_size_t::samples_frequence_min` (C++ member), 220
- `UAC_FREQUENCY_ANY` (C macro), 222
- `uac_input_cb_t` (C++ type), 248
- `uac_mic_streaming_read` (C++ function), 217
- `uac_output_cb_t` (C++ type), 248
- `uac_set_mute_cb_t` (C++ type), 248
- `uac_set_volume_cb_t` (C++ type), 249
- `uac_spk_streaming_write` (C++ function), 216
- `uac_streaming_config` (C++ function), 216
- `UF2_RESET_REASON_VALUE` (C macro), 253
- `uint24_t` (C++ struct), 44
- `uint24_t::u24` (C++ member), 44
- `undervoltage_event_data_t` (C++ struct), 362
- `undervoltage_event_data_t::undervalue` (C++ member), 362
- `undervoltage_event_data_t::voltage_value` (C++ member), 362
- `update_complete_cb_t` (C++ type), 253
- `usb_stream_state_t` (C++ enum), 223
- `usb_stream_state_t::STREAM_CONNECTED` (C++ enumerator), 223
- `usb_stream_state_t::STREAM_DISCONNECTED` (C++ enumerator), 223
- `usb_stream_t` (C++ enum), 223
- `usb_stream_t::STREAM_MAX` (C++ enumerator), 223
- `usb_stream_t::STREAM_UAC_MIC` (C++ enumerator), 223
- `usb_stream_t::STREAM_UAC_SPK` (C++ enumerator), 223
- `usb_stream_t::STREAM_UVC` (C++ enumerator), 223
- `usb_streaming_connect_wait` (C++ function), 216
- `usb_streaming_control` (C++ function), 216
- `usb_streaming_start` (C++ function), 216
- `usb_streaming_state_register` (C++ function), 216
- `usb_streaming_stop` (C++ function), 216
- `usbh_cdc_config` (C++ struct), 235
- `usbh_cdc_config::cbs` (C++ member), 235
- `usbh_cdc_config::itf_num` (C++ member), 235
- `usbh_cdc_config::pid` (C++ member), 235
- `usbh_cdc_config::rx_buffer_size` (C++ member), 235
- `usbh_cdc_config::tx_buffer_size` (C++ member), 235
- `usbh_cdc_config::vid` (C++ member), 235
- `usbh_cdc_create` (C++ function), 231
- `usbh_cdc_delete` (C++ function), 231
- `usbh_cdc_desc_print` (C++ function), 234
- `usbh_cdc_device_config_t` (C++ type), 236
- `usbh_cdc_driver_config_t` (C++ struct), 234
- `usbh_cdc_driver_config_t::new_dev_cb` (C++ member), 234
- `usbh_cdc_driver_config_t::skip_init_usb_host_driver` (C++ member), 234
- `usbh_cdc_driver_config_t::task_coreid` (C++ member), 234
- `usbh_cdc_driver_config_t::task_priority` (C++ member), 234
- `usbh_cdc_driver_config_t::task_stack_size` (C++ member), 234
- `usbh_cdc_driver_config_t::user_data` (C++ member), 234
- `usbh_cdc_driver_install` (C++ function), 230
- `usbh_cdc_driver_uninstall` (C++ function), 231
- `usbh_cdc_event_callbacks_t` (C++ struct), 234
- `usbh_cdc_event_callbacks_t::connect` (C++ member), 235
- `usbh_cdc_event_callbacks_t::disconnect` (C++ member), 235
- `usbh_cdc_event_callbacks_t::notif_cb` (C++ member), 235
- `usbh_cdc_event_callbacks_t::recv_data` (C++ member), 235
- `usbh_cdc_event_callbacks_t::user_data` (C++ member), 235
- `usbh_cdc_event_cb_t` (C++ type), 236
- `usbh_cdc_flush_rx_buffer` (C++ function), 233
- `usbh_cdc_flush_tx_buffer` (C++ function), 233

- usbh_cdc_get_rx_buffer_size (C++ *function*), 233
 - usbh_cdc_get_state (C++ *function*), 234
 - usbh_cdc_handle_t (C++ *type*), 236
 - usbh_cdc_new_dev_cb_t (C++ *type*), 236
 - usbh_cdc_notif_cb_t (C++ *type*), 236
 - usbh_cdc_read_bytes (C++ *function*), 232
 - usbh_cdc_register_new_dev_cb (C++ *function*), 233
 - usbh_cdc_send_custom_request (C++ *function*), 231
 - usbh_cdc_state_t (C++ *enum*), 236
 - usbh_cdc_state_t::USBH_CDC_CLOSE (C++ *enumerator*), 236
 - usbh_cdc_state_t::USBH_CDC_OPEN (C++ *enumerator*), 236
 - usbh_cdc_state_t::USBH_CDC_STATE_MAX (C++ *enumerator*), 236
 - usbh_cdc_unregister_new_dev_cb (C++ *function*), 233
 - usbh_cdc_write_bytes (C++ *function*), 232
 - uvc_config (C++ *struct*), 218
 - uvc_config::ep_addr (C++ *member*), 219
 - uvc_config::ep_mps (C++ *member*), 219
 - uvc_config::flags (C++ *member*), 219
 - uvc_config::format (C++ *member*), 218
 - uvc_config::format_index (C++ *member*), 219
 - uvc_config::frame_buffer (C++ *member*), 218
 - uvc_config::frame_buffer_size (C++ *member*), 218
 - uvc_config::frame_cb (C++ *member*), 218
 - uvc_config::frame_cb_arg (C++ *member*), 218
 - uvc_config::frame_height (C++ *member*), 218
 - uvc_config::frame_index (C++ *member*), 219
 - uvc_config::frame_interval (C++ *member*), 218
 - uvc_config::frame_width (C++ *member*), 218
 - uvc_config::interface (C++ *member*), 219
 - uvc_config::interface_alt (C++ *member*), 219
 - uvc_config::xfer_buffer_a (C++ *member*), 218
 - uvc_config::xfer_buffer_b (C++ *member*), 218
 - uvc_config::xfer_buffer_size (C++ *member*), 218
 - uvc_config::xfer_type (C++ *member*), 219
 - uvc_config_t (C++ *type*), 223
 - uvc_device_config (C++ *function*), 245
 - uvc_device_config_t (C++ *struct*), 246
 - uvc_device_config_t::cb_ctx (C++ *member*), 246
 - uvc_device_config_t::fb_get_cb (C++ *member*), 246
 - uvc_device_config_t::fb_return_cb (C++ *member*), 246
 - uvc_device_config_t::start_cb (C++ *member*), 246
 - uvc_device_config_t::stop_cb (C++ *member*), 246
 - uvc_device_config_t::uvc_buffer (C++ *member*), 246
 - uvc_device_config_t::uvc_buffer_size (C++ *member*), 246
 - uvc_device_deinit (C++ *function*), 245
 - uvc_device_init (C++ *function*), 245
 - uvc_fb_t (C++ *struct*), 245
 - uvc_fb_t::buf (C++ *member*), 245
 - uvc_fb_t::format (C++ *member*), 246
 - uvc_fb_t::height (C++ *member*), 245
 - uvc_fb_t::len (C++ *member*), 245
 - uvc_fb_t::timestamp (C++ *member*), 246
 - uvc_fb_t::width (C++ *member*), 245
 - uvc_format_t (C++ *enum*), 247
 - uvc_format_t::UVC_FORMAT_H264 (C++ *enumerator*), 247
 - uvc_format_t::UVC_FORMAT_JPEG (C++ *enumerator*), 247
 - uvc_frame_size_list_get (C++ *function*), 217
 - uvc_frame_size_reset (C++ *function*), 217
 - uvc_frame_size_t (C++ *struct*), 219
 - uvc_frame_size_t::height (C++ *member*), 220
 - uvc_frame_size_t::interval (C++ *member*), 220
 - uvc_frame_size_t::interval_max (C++ *member*), 220
 - uvc_frame_size_t::interval_min (C++ *member*), 220
 - uvc_frame_size_t::interval_step (C++ *member*), 220
 - uvc_frame_size_t::width (C++ *member*), 220
 - uvc_input_fb_get_cb_t (C++ *type*), 246
 - uvc_input_fb_return_cb_t (C++ *type*), 246
 - uvc_input_start_cb_t (C++ *type*), 246
 - uvc_input_stop_cb_t (C++ *type*), 246
 - uvc_streaming_config (C++ *function*), 215
 - uvc_xfer_t (C++ *enum*), 223
 - uvc_xfer_t::UVC_XFER_BULK (C++ *enumerator*), 223
 - uvc_xfer_t::UVC_XFER_ISOC (C++ *enumerator*), 223
 - uvc_xfer_t::UVC_XFER_UNKNOWN (C++ *enumerator*), 223
- ## V
- video_frame_format (C++ *enum*), 269
 - video_frame_format::FORMAT_H264 (C++ *enumerator*), 269
 - video_frame_format::FORMAT_MJPEG (C++ *enumerator*), 269
 - video_frame_info_t (C++ *struct*), 267

video_frame_info_t::frame_format (C++ member), 267
 video_frame_info_t::height (C++ member), 267
 video_frame_info_t::width (C++ member), 267
 video_write_cb (C++ type), 269

Z

ZERO_CROSS_ADVANCE (C macro), 413
 zero_cross_cb_t (C++ type), 469
 ZERO_CROSS_DETECTION_ACCURACY (C macro), 413
 zero_detect_cb_param_t (C++ union), 466
 zero_detect_cb_param_t::signal_falling_edge_event_data_t (C++ member), 466
 zero_detect_cb_param_t::signal_falling_edge_event_data_t (C++ struct), 467
 zero_detect_cb_param_t::signal_falling_edge_event_data_t::full_cycle_us (C++ member), 467
 zero_detect_cb_param_t::signal_falling_edge_event_data_t::invalid_count (C++ member), 467
 zero_detect_cb_param_t::signal_falling_edge_event_data_t::valid_count (C++ member), 467
 zero_detect_cb_param_t::signal_freq_event_data_t (C++ member), 466
 zero_detect_cb_param_t::signal_freq_event_data_t (C++ struct), 467
 zero_detect_cb_param_t::signal_freq_event_data_t::cap_edge (C++ member), 467
 zero_detect_cb_param_t::signal_freq_event_data_t::full_cycle_us (C++ member), 467
 zero_detect_cb_param_t::signal_invalid_event_data_t (C++ member), 466
 zero_detect_cb_param_t::signal_invalid_event_data_t (C++ struct), 467
 zero_detect_cb_param_t::signal_invalid_event_data_t::cap_edge (C++ member), 467
 zero_detect_cb_param_t::signal_invalid_event_data_t::full_cycle_us (C++ member), 467
 zero_detect_cb_param_t::signal_invalid_event_data_t::invalid_count (C++ member), 467
 zero_detect_cb_param_t::signal_rising_edge_event_data_t (C++ member), 466
 zero_detect_cb_param_t::signal_rising_edge_event_data_t (C++ struct), 467
 zero_detect_cb_param_t::signal_rising_edge_event_data_t::full_cycle_us (C++ member), 468
 zero_detect_cb_param_t::signal_rising_edge_event_data_t::invalid_count (C++ member), 468
 zero_detect_cb_param_t::signal_rising_edge_event_data_t::valid_count (C++ member), 468
 zero_detect_cb_param_t::signal_valid_event_data_t (C++ member), 466
 zero_detect_cb_param_t::signal_valid_event_data_t (C++ struct), 468
 zero_detect_cb_param_t::signal_valid_event_data_t::cap_edge (C++ member), 468

zero_detect_cb_param_t::signal_valid_event_data_t (C++ member), 468
 zero_detect_config_t (C++ struct), 468
 zero_detect_config_t::capture_pin (C++ member), 468
 zero_detect_config_t::freq_range_max_hz (C++ member), 469
 zero_detect_config_t::freq_range_min_hz (C++ member), 469
 zero_detect_config_t::invalid_times (C++ member), 468
 zero_detect_config_t::signal_lost_time_us (C++ member), 468
 zero_detect_config_t::valid_times (C++ member), 468
 zero_detect_config_t::zero_driver_type (C++ member), 468
 zero_detect_config_t::zero_full_cycle_us (C++ member), 468
 zero_detect_config_t::zero_signal_type (C++ member), 468
 zero_detect_create (C++ function), 465
 zero_detect_delete (C++ function), 465
 zero_detect_event_t (C++ enum), 469
 zero_detect_event_t::SIGNAL_FALLING_EDGE (C++ enumerator), 469
 zero_detect_event_t::SIGNAL_FREQ_OUT_OF_RANGE (C++ enumerator), 469
 zero_detect_event_t::SIGNAL_INVALID (C++ enumerator), 469
 zero_detect_event_t::SIGNAL_LOST (C++ enumerator), 469
 zero_detect_event_t::SIGNAL_RISING_EDGE (C++ enumerator), 469
 zero_detect_event_t::SIGNAL_VALID (C++ enumerator), 469
 zero_detect_get_power_status (C++ function), 465
 zero_detect_get_signal_type (C++ function), 466
 zero_detect_handle_t (C++ type), 469
 zero_detect_pause (C++ function), 465
 zero_detect_register_cb (C++ function), 466
 zero_detect_resume (C++ function), 465
 zero_detect_signal_invalid_status (C++ function), 466
 ZERO_DETECTION_INIT_CONFIG_DEFAULT (C macro), 469
 zero_driver_type_t (C++ enum), 470
 zero_driver_type_t::GPIO_TYPE (C++ enumerator), 470
 zero_driver_type_t::valid_count (C++ member), 470
 ZERO_REPEAT_TIME (C macro), 413
 zero_show_data (C++ function), 465
 zero_signal_edge_t (C++ enum), 470
 zero_signal_edge_t::CAP_EDGE_NEG (C++ enumerator), 470
 zero_signal_edge_t::CAP_EDGE_POS (C++ enumerator), 470

`zero_signal_type_t` (C++ *enum*), [469](#)
`zero_signal_type_t::PULSE_WAVE` (C++
enumerator), [469](#)
`zero_signal_type_t::SQUARE_WAVE` (C++
enumerator), [469](#)
`ZERO_STABLE_FLAG_CNT` (C *macro*), [413](#)