ESP-IoT-Solution

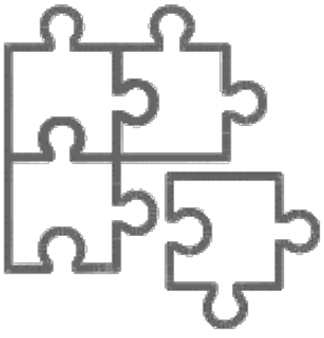
Jun 28, 2023

CONTENTS

[??]

This is the documentation for ESP-IoT-Solution Development Framework.

ESP-IoT-Solution contains device drivers and code frameworks for the development of IoT system, which works as extra components of ESP-IDF and much easier to start.

		
Get Started	Display	USB Host&Device
		
GUI	Input	Sensors
		
Audio	Security&Encryption	Contribute

CHAPTER

ONE

GET STARTED

[??]

This document is intended to help you set up the development environment for ESP-IoT-Solution (Espressif IoT Solution). After that, a simple example will show you how to use ESP-IoT-Solution to set up environment, create a project, build and flash firmware onto an ESP series board, etc.

1.1 ESP-IoT-Solution Introduction

ESP-IoT-Solution contains peripheral drivers and code frameworks commonly used in IoT system development, which provide complementary components to ESP-IDF to facilitate simpler development, mainly including the following contents:

- Device drivers such as sensors, screens, audio devices, input devices, actuators, and etc.
- Code framework and related documents of low power management, security encryption, storage and etc.
- Entrance guideline for Espressif's open-source solutions from the perspective of practical application.

1.1.1 ESP-IoT-Solution Versions

Specifications of different ESP-IoT-Solution versions are listed in the following table:

ESP-IoT-Solution version	Corresponding ESP-IDF version	Main changes	Status
master	>=v4.4	support component manager and new chips	New features development branch
release/v1.1	v4.0.1	IDF version updated, deleted codes that have been moved to other repos	Stop maintenance
release/v1.0	v3.2.2	Legacy version	Stop maintenance

Since the `master` branch uses the `ESP Component Manager` to manager components, each of them is a separate package, and each package may support a different version of the ESP-IDF, which will be declared in the component's `idf_component.yml` file.

1.2 ESP-IDF Introduction

ESP-IDF is the IoT development framework for ESP series SoCs provided by Espressif, including:

- A series of libraries and header files, providing core components required for building software projects based on ESP SoC;
- Common tools and functions used during the development and manufacturing processes, e.g., build, flashing, debugging, measurement and etc.

Note: For detailed information, please go to ESP-IDF Programming Guide.

1.3 ESP Series SoC Introduction

You can select any development board from ESP series to get started with ESP-IoT-Solution, or select a supported board from the Boards Component directly for a quick start.

ESP series SoC support the following features:

- 2.4 GHz Wi-Fi
- Bluetooth
- High-performance single core, dual-core processor, capable of running at 240 MHz
- Ultra-low-power co-processor
- Various peripherals including GPIO, I2C, I2S, SPI ,UART, SDIO, RMT, LEDC PWM, Ethernet, TWAI®, Touch, USB OTG and etc.
- Rich memory resources, including up to 520 KB internal RAM and can support external PSRAM
- Support security functions, e.g., hardware encryption

ESP series of SoC are designed with the 40nm technology, showing the best power and RF performance, versatility and reliability in a wide variety of application and power scenarios.

Note: The configuration varies by SoC series, please refer to ESP Product Selector for details.

1.4 Setting up Development Environment

1.4.1 1. Get ESP-IDF

As ESP-IoT-Solution relies on ESP-IDF basic functions and build tools, please set up ESP-IDF development environment first following ESP-IDF Installation Step by Step. Please note that different versions of ESP-IoT-Solution may rely on different ESP-IDF versions, please refer to *ESP-IoT-Solution Versions* for specifications.

1.4.2 2. Get ESP-IoT-Solution

For `master` version, please use the following command:

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

For `release/v1.1` version, please use the following command:

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

For other versions, please also use this command with `release/v1.1` replaced by your target branch name.

1.5 Use ESP-IoT-Solution Components

If you just want to use the components in ESP-IoT-Solution, we recommend you use it from the ESP Component Registry.

The registered components in ESP-IoT-Solution are listed in README.md , You can directly add the components from the Component Registry to your project by using the `idf.py add-dependency` command under your project's root directory. eg run `idf.py add-dependency "espressif/usb_stream"` to add the `usb_stream`, the component will be downloaded automatically during the `CMake` step.

Please refer to IDF Component Manager for details.

1.6 Build and Download

1.6.1 1. Set up the environment variables

The tools installed in above steps are not yet added to the PATH environment variables. To make the tools usable from the command line, please follow the following steps to add environment variables:

- Add ESP-IDF environment variables:

  For Windows system, please open the Command Prompt and run:

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

  For Linux and macOS, please run:

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

  Please remember to replace the paths in above commands as your actual paths.

- Add IOT_SOLUTION_PATH environment variables:

  For Windows system, please open the Command Prompt and run:

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

  For Linux and macOS, please run:

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

Note: The environment variables set by the above method are only valid in the current terminal. Please repeat above steps if you open a new terminal.

1.6.2 2. Set build target

ESP-IDF supports multiple chips as `esp32`, `esp32s2` and others, please set your target chip before building (the default target is `esp32`). For example, you can set the build target as `esp32s2`.

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

For examples in ESP-IoT-Solution developed based on Boards Component, you can go to `Board Options -> Choose Target Board` in `menuconfig` to choose a target board:

```
idf.py menuconfig
```

1.6.3 3. Build and download the program

Use the `idf.py` tool to build and download the program with:

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

Please replace PORT with your board's port name. Serial ports have the following patterns in their names: Windows is like `COMx`; Linux starting with `/dev/ttyUSBx`; macOS usually is `/dev/cu.`.

1.6.4 4. Serial print log

Use the `idf.py` tool to see logs:

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

Do not forget to replace PORT with your serial port name (`COMx` for Windows; `/dev/ttyUSBx` for Linux; `/dev/cu.` for macOS).

1.7 Related Documents

- ESP-IDF Installation Step by Step
- ESP-IDF Get Started
- ESP Product Selector

CHAPTER

TWO

BASIC COMPONENT

[??]

2.1 Communication Bus

[??]

The communication bus component (Bus) is a set of application-layer code built on top of the ESP-IDF peripheral driver code, including `i2c_bus`, `spi_bus` and etc. It is mainly used for bus communication between ESP chips and external devices. From the point of application development, this component has the following features:

1. Simplified peripheral initialization processes
2. Thread-safe device operations
3. Simple and flexible RW operations

This component abstracts the following concepts:

1. Bus: the resource and configuration option shared between devices during communication
2. Device: device specific resource and configuration option during communication

Each physical peripheral bus can mount one or more devices if the electrical condition allows, with the SPI bus addressing devices based on CS pins and the I2C bus addressing devices based on their addresses, thus achieving software independence between different devices on the same bus.

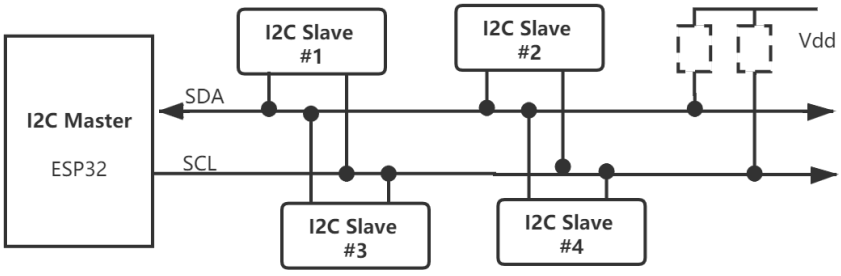


Fig. 1: i2c_bus Connection Diagram

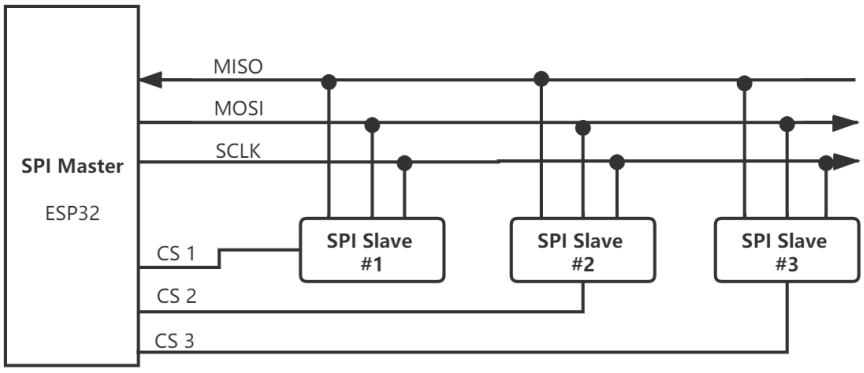


Fig. 2: spi_bus Connection Diagram

2.1.1 How to Use i2c_bus

1. Create a bus: create a bus object using *i2c_bus_create()*. During the process, you need to specify the I2C port number and the bus configuration option `i2c_config_t`, which includes SDA and SCL pin numbers, pull-up and pull-down modes, as these are determined when the system is designed and normally will not be changed at runtime. The bus configuration option also includes the default clock frequency of the bus, which is used when the device does not specify a frequency.
2. Create a device: use *i2c_bus_device_create()* to create a device on the bus object created in the first step. During the process, you need to specify bus handle, the I2C address of the device, and the clock frequency when the device is running. The frequency will be dynamically changed during I2C transmission based on device configuration options. The device clock frequency can be configured as 0, indicating the current bus frequency is used by default.
3. Data reading: use *i2c_bus_read_byte()* or *i2c_bus_read_bytes()* to read `Byte` data; use *i2c_bus_read_bit()* or *i2c_bus_read_bits()* to read `bit` data. During the process, you only need to pass in the device handle, the device register address, a buffer to hold the read data, the read length, etc. The register address can be configured as `NULL_I2C_MEM_ADDR` for devices without internal registers.
4. Data writing: use *i2c_bus_write_byte()* or *i2c_bus_write_bytes()* to write `Byte` data; use *i2c_bus_write_bit()* or *i2c_bus_write_bits()* to write `bit` data. During the process, you only need to pass in the device handle, the device register address, the data location to be written, the write length, etc. The register address can be configured as `NULL_I2C_MEM_ADDR` for devices without internal registers.
5. Delete device and bus: if all i2c_bus communication has been completed, you can free your system resources by deleting devices and bus objects. Use *i2c_bus_device_delete()* to delete created devices respectively, then use *i2c_bus_delete()* to delete bus resources. If the bus is deleted with the device not being deleted yet, this operation will not take effect.

Example:

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

(continues on next page)

(continued from previous page)

```
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
```

Note: For some special application scenarios:

1. When the address of a register is 16-bit, you can use *i2c_bus_read_reg16()* or *i2c_bus_write_reg16()* to read or write its data;
2. For devices that need to skip the address phase or need to add a command phase, you can operate using *i2c_bus_cmd_begin()* combined with I2C command link.

2.1.2 How to Use spi_bus

1. Create a bus: use *spi_bus_create()* to create a bus object. During the process, you need to specify the SPI port number (can choose between `SPI2_HOST` and `SPI3_HOST`) and the bus configuration option `spi_config_t`, which includes the pin numbers of `MOSI`, `MISO` and `SCLK`, as these are determined when the system is designed and normally will not be changed at runtime. The bus configuration option also includes `max_transfer_sz` to configure the maximum data size during a transmission. When `max_transfer_sz` is configured to 0, it means the maximum size will be the default value 4096.
2. Create a device: use *spi_bus_device_create()* to create a device on the bus object created in the first step. During the process, you need to specify the bus handle, the `CS` pin number of the device, device operation mode, the clock frequency when the device is running. The device mode and frequency will be dynamically changed during SPI transmissions based on device configuration options.
3. Data transmission: use *spi_bus_transfer_byte()*, *spi_bus_transfer_bytes()*, *spi_bus_transfer_reg16()* or *spi_bus_transfer_reg32()* to transfer data directly. Data send and receive can be operated at the same time since SPI communication is a full-duplex communication. During the process, you only need to pass in the device handle, data to be transmitted, a buffer to hold the read data, transmission length, etc.
4. Delete device and bus: if all spi_bus communication has been completed, you can free your system resources by deleting devices and bus objects. Use *spi_bus_device_delete()* to delete created devices respectively, then use *spi_bus_delete()* to delete bus resources. If the bus is deleted with the device not being deleted yet, this operation will not take effect.

Example:

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

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

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

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

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

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

Note: For some special application scenarios, you can operate using *spi_bus_transmit_begin()* combined with spi_transaction_t directly.

2.1.3 Adapted IDF Versions

- ESP-IDF v4.0 and later versions.

2.1.4 Supported Chips

- ESP32
- ESP32-S2

2.1.5 API Reference

i2c_bus API Reference

Header File

- 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.

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

Parameters

- `port`: I2C port number
- `conf`: Pointer to I2C bus configuration

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

Delete and release the I2C bus resource.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `p_bus_handle`: Point to the I2C bus handle, if delete succeed handle will set to NULL.

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

Scan i2c devices attached on i2c bus.

Return uint8_t Total number of devices found on the I2C bus

Parameters

- `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, higer addresses will be discarded if num less-than the total number found on the I2C bus.

uint32_t **i2c_bus_get_current_clk_speed**(*i2c_bus_handle_t bus_handle*)
Get current active clock speed.

Return uint32_t current clock speed

Parameters

- `bus_handle`: I2C bus handle

uint8_t **i2c_bus_get_created_device_num**(*i2c_bus_handle_t bus_handle*)
Get created device number of the bus.

Return uint8_t created device number of the bus

Parameters

- `bus_handle`: I2C bus handle

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.

Return i2c_bus_device_handle_t return a device handle if created successfully, return NULL if failed.

Parameters

- `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.

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.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `p_dev_handle`: Point to the I2C device handle, if delete succeed handle will set to NULL.

uint8_t **i2c_bus_device_get_address**(*i2c_bus_device_handle_t dev_handle*)
Get device's I2C address.

Return uint8_t I2C address, return NULL_I2C_DEV_ADDR if dev_handle is invalid.

Parameters

- `dev_handle`: I2C device handle

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.

Return 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.

Parameters

- `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

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.

Return 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.

Parameters

- `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

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.

Return 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.

Parameters

- `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.

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.

Return 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.

Parameters

- `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

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.

Return 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.

Parameters

- `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.

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.

Return 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.

Parameters

- `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.

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.

Return 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.

Parameters

- `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.

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.

Return 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.

Parameters

- `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.

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.

Note Only call this function when `i2c_bus_read/write_xx` do not meet the requirements

Return 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.

Parameters

- `dev_handle`: I2C device handle
- `cmd`: I2C command handler

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

Return 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.

Parameters

- `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.

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

Return 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.

Parameters

- `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

Macros

NULL_I2C_MEM_ADDR

set mem_address to NULL_I2C_MEM_ADDR if i2c device has no internal address during read/write

NULL_I2C_DEV_ADDR

invalid i2c device address

gpio_pad_select_gpio

portTICK_RATE_MS

Type Definitions

typedef void ***i2c_bus_handle_t**

i2c bus handle

typedef void ***i2c_bus_device_handle_t**

i2c device handle

spi_bus API Reference

Header File

- 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.

Return spi_bus_handle_t handle for spi bus operation, NULL if failed.

Parameters

- `host_id`: SPI peripheral that controls this bus, SPI2_HOST or SPI3_HOST
- `bus_conf`: spi bus configurations details in *spi_config_t*

esp_err_t **spi_bus_delete**(*spi_bus_handle_t *p_bus_handle*)
Deinitialize and delete the spi bus.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_FAIL Fail
- ESP_OK Success

Parameters

- `p_bus_handle`: pointer to spi bus handle, if delete succeed handle will set to NULL.

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.

Return spi_bus_device_handle_t handle for device operation, NULL if failed.

Parameters

- `bus_handle`: handle for spi bus operation.
- `device_conf`: spi device configurations details in *spi_device_config_t*

esp_err_t **spi_bus_device_delete**(*spi_bus_device_handle_t *p_dev_handle*)
Deinitialize and remove the device from spi bus.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_FAIL Fail
- ESP_OK Success

Parameters

- `p_dev_handle`: pointer to device handle, if delete succeed handle will set to NULL.

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.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid

- ESP_ERR_TIMEOUT if bus is busy
- ESP_OK on success

Parameters

- `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.

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.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if bus is busy
- ESP_OK on success

Parameters

- `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.

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.

Note Only call this function when `spi_bus_transfer_xx` do not meet the requirements

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if bus is busy
- ESP_OK on success

Parameters

- `dev_handle`: handle for device operation.
- `p_trans`: Description of transaction to execute

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.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if bus is busy
- ESP_OK on success

Parameters

- `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.

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.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if bus is busy
- ESP_OK on success

Parameters

- `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.

Structures

struct spi_config_t

spi bus initialization parameters.

Public Members

gpio_num_t **miso_io_num**

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

gpio_num_t **mosi_io_num**

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

gpio_num_t **sclk_io_num**

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

int **max_transfer_sz**

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

struct spi_device_config_t

spi device initialization parameters.

Public Members

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

uint8_t **mode**
modes (0,1,2,3) that correspond to the four possible clocking configurations

int **clock_speed_hz**
spi clock speed, divisors of 80MHz, in Hz. See `SPI_MASTER_FREQ_*

Macros

NULL_SPI_CS_PIN
set cs_io_num to NULL_SPI_CS_PIN if spi device has no CP pin

Type Definitions

typedef void ***spi_bus_handle_t**
spi bus handle

typedef void ***spi_bus_device_handle_t**
spi device handle

2.2 I2S LCD ??

2.2.1 API ??

Header File

- bus/include/i2s_lcd_driver.h

Functions

i2s_lcd_handle_t **i2s_lcd_driver_init**(**const** *i2s_lcd_config_t* **config*)
Initilize i2s lcd driver.

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

Parameters

- `config`: configuration of i2s

esp_err_t **i2s_lcd_driver_deinit**(*i2s_lcd_handle_t handle*)
Deinit i2s lcd driver.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG handle is invalid

Parameters

- `handle`: i2s lcd driver handle to deinitilize

esp_err_t **i2s_lcd_write_data**(*i2s_lcd_handle_t handle*, uint16_t *data*)
Write a data to LCD.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG handle is invalid

Parameters

- `handle`: i2s lcd driver handle
- `data`: Data to write

esp_err_t **i2s_lcd_write_cmd**(*i2s_lcd_handle_t handle*, uint16_t *cmd*)
Write a command to LCD.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG handle is invalid

Parameters

- `handle`: Handle of i2s lcd driver
- `cmd`: command to write

esp_err_t **i2s_lcd_write_command**(*i2s_lcd_handle_t handle*, **const** uint8_t **cmd*, uint32_t *length*)
Write a command to LCD.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG handle is invalid

Parameters

- `handle`: Handle of i2s lcd driver
- `cmd`: command to write
- `length`: length of command

esp_err_t **i2s_lcd_write**(*i2s_lcd_handle_t handle*, **const** uint8_t **data*, uint32_t *length*)
Write block data to LCD.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG handle is invalid

Parameters

- `handle`: Handle of i2s lcd driver
- `data`: Pointer of data
- `length`: length of data

esp_err_t **i2s_lcd_acquire**(*i2s_lcd_handle_t handle*)
acquire a lock

Return Always return ESP_OK

Parameters

- `handle`: Handle of i2s lcd driver

esp_err_t **i2s_lcd_release**(*i2s_lcd_handle_t handle*)
release a lock

Return Always return ESP_OK

Parameters

- `handle`: Handle of i2s lcd driver

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 Boards Component

[??]

This document mainly introduces the use of a board support component (Boards). As a common component of examples, this component can provide unified pin macro definitions and hardware-independent initialization operations to applications. Applications developed based on this component are compatible with different development boards at the same time with the following features:

1. Provides unified macro definitions for pins
2. Provides default peripheral configuration parameters
3. Provides unified board-level initialization interfaces
4. Provides hardware control interfaces for development boards

The following figure shows the structure of the Boards component:

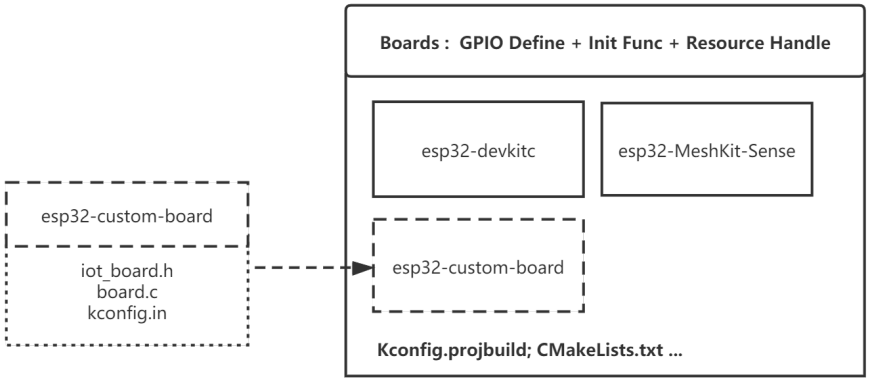


Fig. 3: Boards Component Diagram

- The Boards component contains the following:
  - `board_common.h`, contains the function declaration of the common API;
  - `board_common.c`, contains the function implementation of the common API (weak function);
  - `Kconfig.projbuild`, contains common configuration items;
- The subfolders named after the development board name includes the following:
  - `iot_board.h` provides the gpio definition of the development board, and the board's unique custom API function declaration

- `board.c` provides user implementation of common API (Covering default weak function), custom API function implementation
- `kconfig.in` provides custom configuration items unique to the development board.

Note: The Boards component is provided in `examples/common_components/boards`.

2.3.1 Instructions

1. Initialize development board: use `iot_board_init` in `app_main` to initialize the development board. you can also do some configurations regarding this process using *The Switch and Configuration of a Development Board* in `menuconfig`;
2. Get the handle of a peripheral: use `iot_board_get_handle` and `board_res_id_t` to get peripheral resources. `NULL` will be returned if this peripheral is not initialized;
3. Operate on peripherals with handles directly.

Example:

```
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.
    */
}
```

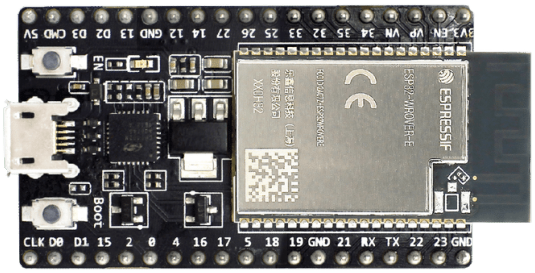
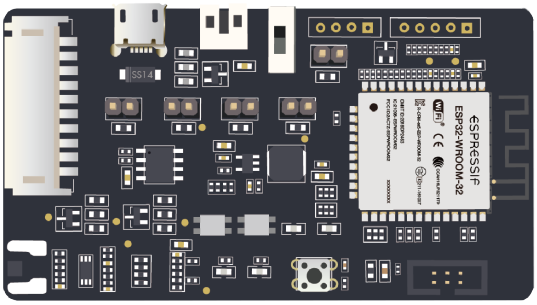
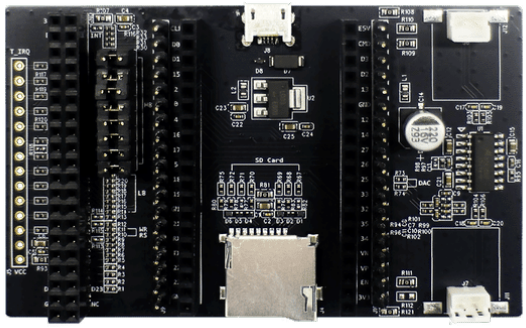
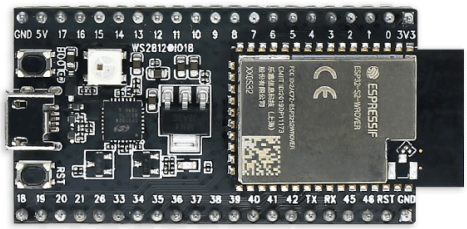
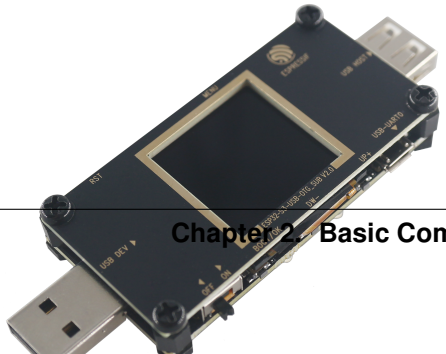
2.3.2 The Switch and Configuration of a Development Board

For applications developed basing on `Boards`, the following steps can be used to switch and configure boards:

1. Select the target development board: select a development board in `menuconfig->Board Options->Choose Target Board`;
2. Configure the development board parameters: `Board Common Options` contains common configurations, such as if enable i2c0 during the initialization of the development board; `XXX Board Options` contains the development board-specific configurations, such as switching the power supply status of the development board, etc.
3. Use `idf.py build flash monitor` to recompile and download the code.

Note: The default target of this build system is `ESP32`, please set the target before building via `idf.py set-target esp32s2` if you need to use `ESP32-S2`.

2.3.3 Supported Development Boards

ESP32 Development Boards	
	
esp32-devkitc	esp32-meshkit-sense
	
esp32-lcdkit	
ESP32-S2 Development Boards	
	
esp32s2-saola	
ESP32-S3 Development Boards	
	

2.3.4 Add a New Development Board

A new development board can be added to quickly adapt to applications developed basing on the `Boards` component.

The main process is as follows:

1. Prepare the necessary `iot_board.h` based on existing example;
2. Add board specific functions or cover the common weak function in `board_xxx.c` according to the requirements;
3. Add configuration options specific to this board in `kconfig.in` according to your needs;
4. Add the information of this board to `Kconfig.projbuild` for users;
5. Add the directory of this board to `CMakeLists.txt` so that it can be indexed by the build system. Please also update `component.mk` if you need to support the old `make` system.

Note: An easy way is to directly copy files of the existing development boards in `Boards` and make simple modifications to add your new board.

2.3.5 Component Dependencies

- Common dependencies: bus, button, led_strip

2.3.6 Adapted IDF Versions

- ESP-IDF v4.4 and later versions.

2.3.7 Supported Chips

- ESP32
- ESP32-S2
- ESP32-S3

CHAPTER

THREE

DISPLAY

[??]

3.1 Screen

[??]

Screen is a very important display device as many information from various applications needs to be displayed to users. Both ESP32 and ESP32-S2 chips support screens driven by I2C interface, 8080 parallel interface, SPI interface and etc. The supported types of screen controllers are listed in the following table:

Controller	Max Resolution	Type
NT35510	480 x 865	Color
ILI9806	480 x 865	Color
RM68120	480 x 865	Color
ILI9486	320 x 480	Color
ILI9341	240 x 320	Color
ST7789	240 x 320	Color
ST7796	320 x 480	Color
SSD1351	128 x 128	Color
SSD1306	128 x 64	Mono
SSD1307	128 x 39	Mono
SSD1322	480 x 128	Gray

Note: The 8080 parallel interface is implemented via the LCD mode in the I2S of ESP32, so sometimes it is called `I2S interface` in this document.

3.1.1 Screen Driver Structure

In order to be more in line with the actual situation where a screen controller has multiple interfaces, the screen driver is divided into two parts: the `interface driver` and the `controller driver`.

- The interface driver: conduct basic reads and writes of commands and data
- The controller driver: display information on screen via interfaces

A controller driver can be designed to switch between different interfaces in hardware level by calling corresponding interface drivers.

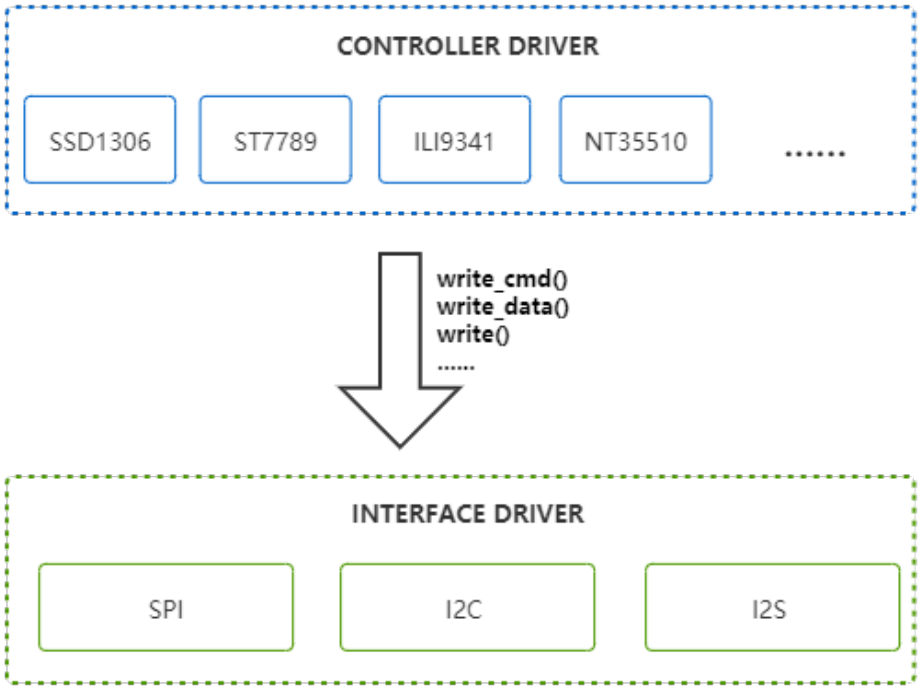


Fig. 1: Screen Driver Structure Diagram

3.1.2 Screen Types

A discussion about screen types will help us to have a clear understanding of drivers. Here, we use colors that can be displayed on the screen to classify screens, rather than the panel material of them such as OLED, LCD and etc. In general, the colors displayed on screen determines the BPP (Bits Per Pixel), and the differences in BPP lead to differences in how the program handles it. Here, we list some ways in which GRAM is mapped to pixel points in below:

From above figures, we can see that there are mainly two types of mapping:

- When BPP >= 8, it is usually a color screen that supports RGB888, RGB666, RGB565 and other codings.
- When BPP < 8, it is usually a mono screen that may either be black-and-white or gray.

When BPP < 8, a byte is mapped to multiple pixels, so a single pixel cannot be controlled directly. In this case, `draw_pixel()` is not supported in the driver, and the parameters of `set_window()` are also limited. When BPP >= 8, each single pixel can be accessed easily.

Attention: For color screens, the driver only supports RGB565 color coding.

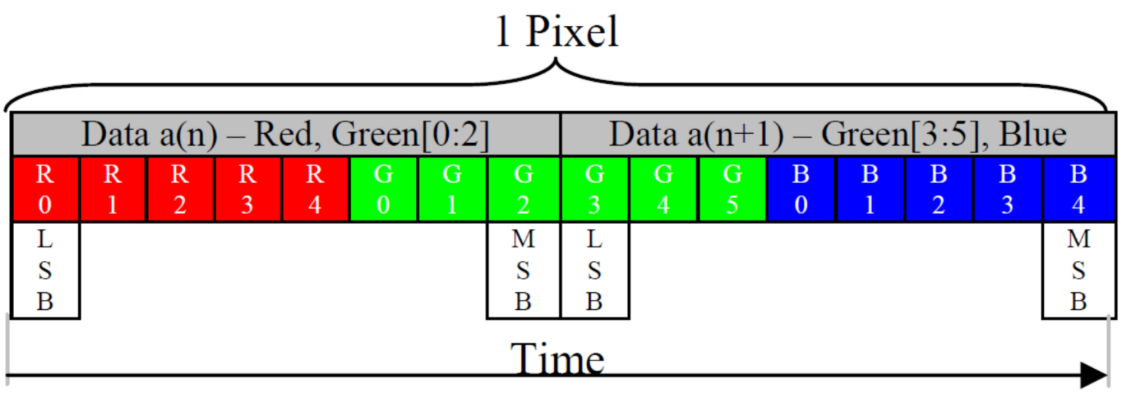


Fig. 2: BPP = 16 GRAM Structure

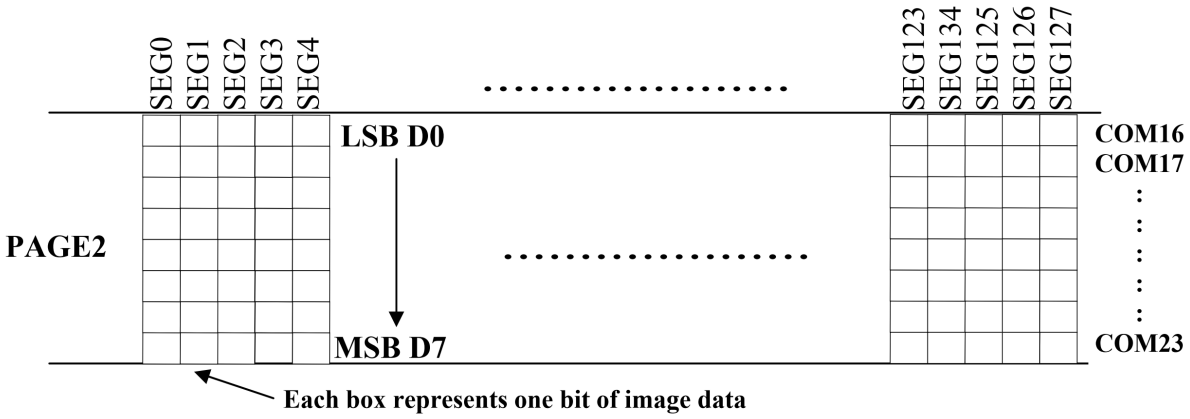


Fig. 3: BPP = 1 GRAM Structure

		SEG0	SEG1	SEG2	SEG3		SEG476	SEG477	SEG478	SEG479	SEG Outputs
		00		00			77		77		RAM Column address (HEX)
COM0	00	D1[3:0]	D1[7:4]	D0[3:0]	D0[7:4]		D239[3:0]	D239[7:4]	D238[3:0]	D238[7:4]	
COM1	01	D241[3:0]	D241[7:4]	D240[3:0]	D240[7:4]		D479[3:0]	D479[7:4]	D478[3:0]	D478[7:4]	
|	|									
COM126	7E	D30241[3:0]	D30241[7:4]	D30240[3:0]	D30240[7:4]		D30479[3:0]	D30479[7:4]	D30478[3:0]	D30478[7:4]	
COM127	7F	D30481[3:0]	D30481[7:4]	D30480[3:0]	D30480[7:4]		D30719[3:0]	D30719[7:4]	D30718[3:0]	D30718[7:4]	
COM Outputs	RAM Row Address (HEX)										

Corresponding to one pixel

Fig. 4: BPP = 4 GRAM Structure

3.1.3 Interface Driver

A screen controller usually has multiple interfaces. On ESP32, three kinds of interfaces as `8080 parallel interface`, `SPI` and `I2C` are typically used to connect to the screen. You can choose one of them as the interface when creating interface drivers via `scr_interface_create()`.

Note: Please remember to select corresponding parameter types when creating different interfaces using `scr_interface_create()`, e.g., select *`i2s_lcd_config_t`* for I2S interface; select *`scr_interface_spi_config_t`* for SPI interface.

To facilitate the use of these interfaces in the driver, all interfaces are defined in display/screen/screen_utility/interface_drv_def.h, which can be called easily by less parameters.

Note: Most screens use big-endian order to store data, while ESP32 uses small-endian mode. You can switch between them in the interface driver you used, based on `swap_data` configurations. **Please note:** when using the SPI interface, the received data **must** be stored in RAM because the IDF's SPI driver itself does not support this swapping function and an additional program in the interface driver will do the work, which require the data to be writable.

3.1.4 Controller Driver

Some common functions of the screen are abstracted using *`scr_driver_t`* in this section according to display and other functions of different screen controllers, in order to port these common functions to different GUI libraries easily. For some non-generic functions of the screen, you need to call its specific functions.

Not all screens have implemented these common functions, since different screen controller has their own functions. For example, for screens with BPP < 8, the function `draw_pixel()` is not supported. And calling an unsupported function will return `ESP_ERR_NOT_SUPPORTED`.

Display Direction

The screen display direction set here is implemented entirely by the screen hardware, and this feature varies from one screen controller to another. There are 8 possible display directions. A display can be rotated by 0°, 90°, 180° or 270° and can also be viewed from the top or bottom, with 0° and top view as its default direction. These 8 (4 × 2) directions can also be represented as a combination of 3 binary switches: X-mirroring, Y-mirroring and X/Y swapping.

The total 8 combinations of display directions are listed in the following table. If the direction of your display is not correct, please check the configuration switches below to make it work properly.

	0 [SCR_DIR_LRTB]		SCR_MIRROR_Y [SCR_DIR_LRBT]
	SCR_MIRROR_X [SCR_DIR_RLTB]		SCR_MIRROR_X| SCR_MIRROR_Y [SCR_DIR_RLBT]
	SCR_SWAP_XY [SCR_DIR_TBLR]		SCR_SWAP_XY| SCR_MIRROR_Y [SCR_DIR_BTLR]
	SCR_SWAP_XY| SCR_MIRROR_X [SCR_DIR_TBRL]		SCR_SWAP_XY| SCR_MIRROR_X| SCR_MIRROR_Y [SCR_DIR_BTRL]

The implementations of display directions are not exactly the same for different screen controllers, and are usually divided into the following cases:

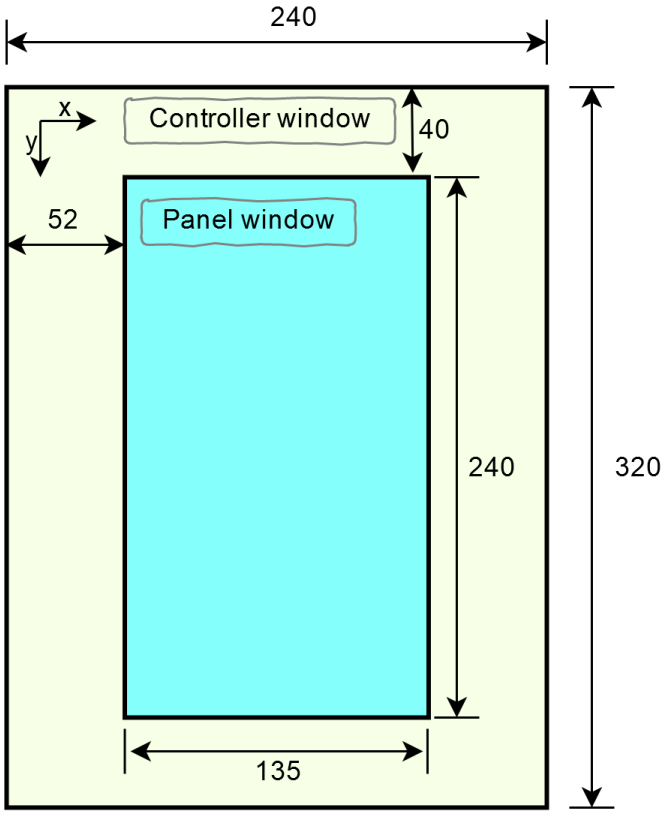
- For color screens, 8 directions are supported.
- For mono screens, e.g., SSD1306, only the first 4 directions defined in `scr_dir_t` are supported, which means they do not support X/Y swapping.

Note: The display direction is also related to the screen panel you used, and you may encounter two types of abnormal cases:

- The display direction is set to `SCR_DIR_LRTB`, but the screen does not show as what listed in the above table. This may be because the alignment on the screen panel is mirrored in the X/Y direction, in which case you need to adjust the rotation to get the desired direction.
- After rotated, the screen does not show anything any more. This may be because the resolution of the screen panel is smaller than that of the screen controller, making the display area not falling completely on the screen panel, in which case you need to set a proper offset for the display area.

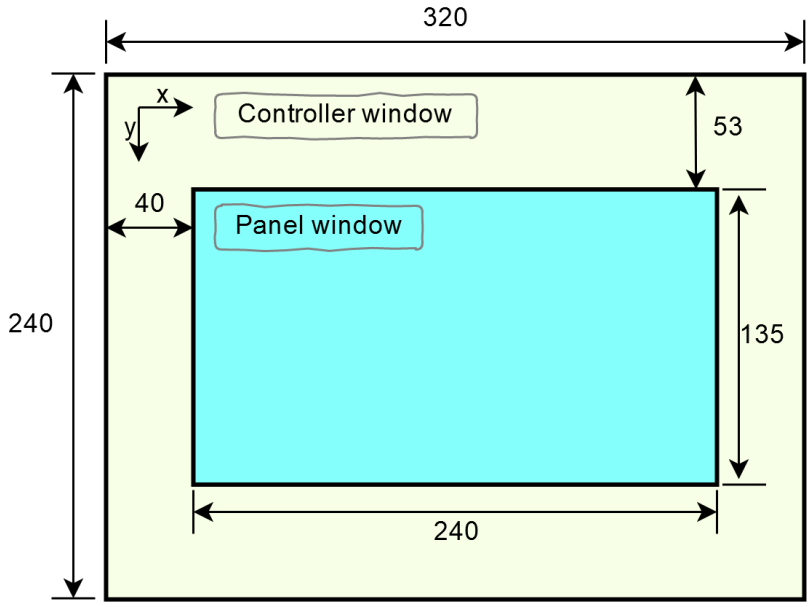
Offset of the Display Area

In some small screens, the resolution of the display area is usually smaller than that of the controller window. Please refer to the following figure:



In this figure, `Controller window` is the window for screen controller, with its resolution as 240 × 320; `Panel window` is the window for screen panel, with its resolution as 135 × 240, which is the display area. From this figure, we can see that the display area is shifted by 52 pixels horizontally and by 40 pixels vertically.

When the screen is rotated 90° anticlockwise, the display area is shifted by 40 pixels horizontally and by 53 pixels vertically, as shown in the figure below:



The screen controller driver will help you to change the offset value automatically according to the rotation of the screen

to maintain a proper display. All you need to do is to properly configure the screen offset and the screen panel size in *scr_controller_config_t* when it is in SCR_DIR_LRTB direction.

Note:

- This only supports screens with BPP >= 8.
- When the resolution of your screen controller is configurable and you find something wrong with the offset, it may be because the selected resolution does not match the actual one, and you should make modifications accordingly, for example, set the ILI9806_RESOLUTION_VER in ili9806.c as the actual resolution for ILI9806.

3.1.5 Application Example

Note: The following examples are no longer maintained. For LCD and LVGL examples, please refer to: i80_controller rgb_panel And spi_lcd_touch

Initialize the Screen

```
scr_driver_t g_lcd; // A screen driver
esp_err_t ret = ESP_OK;

/** Initialize 16bit 8080 interface */
i2s_lcd_config_t i2s_lcd_cfg = {
    .data_width  = 16,
    .pin_data_num = {
        1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
    },
    .pin_num_cs = 45,
    .pin_num_wr = 34,
    .pin_num_rs = 33,
    .clk_freq = 20000000,
    .i2s_port = I2S_NUM_0,
    .buffer_size = 32000,
    .swap_data = false,
};
scr_interface_driver_t *iface_drv;
scr_interface_create(SCREEN_IFACE_8080, &i2s_lcd_cfg, &iface_drv);

/** Find screen driver for ILI9806 */
ret = scr_find_driver(SCREEN_CONTROLLER_ILI9806, &g_lcd);
if (ESP_OK != ret) {
    ESP_LOGE(TAG, "screen find failed");
    return;
}

/** Configure screen controller */
scr_controller_config_t lcd_cfg = {
    .interface_drv = iface_drv,
    .pin_num_rst = -1,      // The reset pin is not connected
    .pin_num_bckl = -1,     // The backlight pin is not connected
    .rst_active_level = 0,
```

(continues on next page)

(continued from previous page)

```
    .bckl_active_level = 1,
    .offset_hor = 0,
    .offset_ver = 0,
    .width = 480,
    .height = 854,
    .rotate = SCR_DIR_LRBT,
};

/** Initialize ILI9806 screen */
g_lcd.init(&lcd_cfg);
```

Note: By default, only the driver of ILI9341 screen is enabled. If you need to use other drivers, please go to `menuconfig -> Component config -> LCD Drivers -> Select Screen Controller` to enable the corresponding screen drivers.

Display Images

```
/** Draw a red point at position (10, 20) */
lcd.draw_pixel(10, 20, COLOR_RED);

/** Draw a bitmap */
lcd.draw_bitmap(0, 0, width_of_pic, height_of_pic, pic_data);
```

Obtain Screen Information

```
scr_info_t lcd_info;
lcd.get_info(&lcd_info);
ESP_LOGI(TAG, "Screen name:%s | width:%d | height:%d", lcd_info.name, lcd_info.width,␣
↪lcd_info.height);
```

3.1.6 API Reference

Header File

- display/screen/screen_driver.h

Functions

esp_err_t **scr_find_driver** (*scr_controller_t controller*, *scr_driver_t *out_screen*)

Find a screen driver.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG Arguments is NULL.
- ESP_ERR_NOT_FOUND Screen controller was not found.

Parameters

- `controller`: Screen controller to initialize
- `out_screen`: Pointer to a screen driver

Structures

struct scr_controller_config_t
configuration of screen controller

Public Members

scr_interface_driver_t ***interface_drv**
Interface driver for screen

int8_t **pin_num_rst**
Pin to hardreset LCD

int8_t **pin_num_bckl**
Pin for control backlight

uint8_t **rst_active_level**
Reset pin active level

uint8_t **bckl_active_level**
Backlight active level

uint16_t **width**
Screen width

uint16_t **height**
Screen height

uint16_t **offset_hor**
Offset of horizontal

uint16_t **offset_ver**
Offset of vertical

scr_dir_t **rotate**
Screen rotate direction

struct scr_info_t
Information of screen.

Public Members

uint16_t **width**
Current screen width, it may change when apply to rotate

uint16_t **height**
Current screen height, it may change when apply to rotate

scr_dir_t **dir**
Current screen direction

scr_color_type_t **color_type**
Color type of the screen, See scr_color_type_t struct

uint8_t **bpp**
Bits per pixel

const char ***name**
Name of the screen

struct scr_driver_t
Define a screen common function.

Public Members

esp_err_t (***init**)(**const** *scr_controller_config_t* *lcd_conf)
Initialize screen.

Return

- ESP_OK on success
- ESP_FAIL Driver not installed

Parameters

- `lcd_conf`: Pointer to a structure with lcd config arguments. see struct *scr_controller_config_t*

esp_err_t (***deinit**)(void)
Deinitialize screen.

Return

- ESP_OK on success
- ESP_FAIL Deinitialize failed
- ESP_ERR_NOT_SUPPORTED unsupported

esp_err_t (***set_direction**)(*scr_dir_t* dir)
Set screen direction of rotation.

Note Not all screens support eight directions, it depends on the screen controller.

Return

- ESP_OK on success
- ESP_FAIL Failed

Parameters

- `dir`: Pointer to a scr_dir_t structure. You can set the direction in two ways, for example, set it to "SCR_DIR_LRBT" or "SCR_MIRROR_Y", They are the same, depending on which expression you want to use

esp_err_t (***set_window**)(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
Set screen window.

Note When the BPP of the screen controller is less than 8, the coordinate value is limited to a multiple of some number

Return

- ESP_OK on success

- ESP_FAIL Failed

Parameters

- `x0`: Starting point in X direction
- `y0`: Starting point in Y direction
- `x1`: End point in X direction
- `y1`: End point in Y direction

esp_err_t (*`write_ram_data`) (uint16_t color)
Write a RAM data.

Return

- ESP_OK on success
- ESP_FAIL Failed

Parameters

- `color`: New color of a pixel

esp_err_t (*`draw_pixel`) (uint16_t x, uint16_t y, uint16_t color)
Draw one pixel in screen with color.

Return

- ESP_OK on success
- ESP_FAIL Failed

Parameters

- `x`: X co-ordinate of set orientation
- `y`: Y co-ordinate of set orientation
- `color`: New color of the pixel

esp_err_t (*`draw_bitmap`) (uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t *bitmap)
Fill the pixels on LCD screen with bitmap.

Return

- ESP_OK on success
- ESP_FAIL Failed

Parameters

- `x`: Starting point in X direction
- `y`: Starting point in Y direction
- `w`: width of image in bitmap array
- `h`: height of image in bitmap array
- `bitmap`: pointer to bitmap array

esp_err_t (*`get_info`) (*scr_info_t* *info)
Get screen information.

Return

- ESP_OK on success
- ESP_FAIL Failed

Parameters

- `info`: Pointer to a *scr_info_t* structure.

Macros

COLOR_BLACK

COLOR_NAVY

COLOR_DARKGREEN

COLOR_DARKCYAN

COLOR_MAROON

COLOR_PURPLE

COLOR_OLIVE

COLOR_LIGHTGREY

COLOR_DARKGREY

COLOR_BLUE

COLOR_GREEN

COLOR_CYAN

COLOR_RED

COLOR_MAGENTA

COLOR_YELLOW

COLOR_WHITE

COLOR_ORANGE

COLOR_GREENYELLOW

COLOR_PINK

COLOR_SILVER

COLOR_GRAY

COLOR_LIME

COLOR_TEAL

COLOR_FUCHSIA

COLOR_ESP_BKGD

Enumerations

enum scr_dir_t
Define all screen direction.

Values:

SCR_DIR_LRTB
From left to right then from top to bottom, this consider as the original direction of the screen

SCR_DIR_LRBT
From left to right then from bottom to top

SCR_DIR_RLTB
From right to left then from top to bottom

SCR_DIR_RLBT
From right to left then from bottom to top

SCR_DIR_TBLR
From top to bottom then from left to right

SCR_DIR_BTLR
From bottom to top then from left to right

SCR_DIR_TBRL
From top to bottom then from right to left

SCR_DIR_BTRL
From bottom to top then from right to left

SCR_DIR_MAX

SCR_MIRROR_X = 0x40
Mirror X-axis

SCR_MIRROR_Y = 0x20
Mirror Y-axis

SCR_SWAP_XY = 0x80
Swap XY axis

enum scr_color_type_t
The types of colors that can be displayed on the screen.

Values:

SCR_COLOR_TYPE_MONO
The screen is monochrome

SCR_COLOR_TYPE_GRAY
The screen is gray

SCR_COLOR_TYPE_RGB565
The screen is colorful

enum scr_controller_t
All supported screen controllers.

Values:

SCREEN_CONTROLLER_ILI9341

SCREEN_CONTROLLER_ILI9342

SCREEN_CONTROLLER_ILI9806

SCREEN_CONTROLLER_ILI9486

SCREEN_CONTROLLER_ILI9488

SCREEN_CONTROLLER_NT35510

SCREEN_CONTROLLER_RM68120

SCREEN_CONTROLLER_ST7789

SCREEN_CONTROLLER_ST7796

SCREEN_CONTROLLER_SSD1351

SCREEN_CONTROLLER_SSD1963

SCREEN_CONTROLLER_SSD1306

SCREEN_CONTROLLER_SSD1307

SCREEN_CONTROLLER_SSD1322

Header File

- display/screen/interface_driver/scr_interface_driver.h

Functions

esp_err_t **scr_interface_create**(*scr_interface_type_t type*, void **config*, *scr_interface_driver_t **out_driver*)

Create screen interface driver.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG Arguments is NULL.
- ESP_FAIL Initialize failed
- ESP_ERR_NO_MEM: Cannot allocate memory.

Parameters

- `type`: Type of screen interface
- `config`: configuration of interface driver
- `out_driver`: Pointer to a screen interface driver

esp_err_t **scr_interface_delete**(**const** *scr_interface_driver_t *driver*)

Delete screen interface driver.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG Arguments is NULL.

Parameters

- `driver`: screen interface driver to delete

Structures

struct scr_interface_spi_config_t
SPI interface configuration.

Public Members

spi_bus_handle_t **spi_bus**
Handle of spi bus

int8_t **pin_num_cs**
SPI Chip Select Pin

int8_t **pin_num_dc**
Pin to select Data or Command for LCD

int **clk_freq**
SPI clock frequency

bool **swap_data**
Whether to swap data

struct scr_interface_i2c_config_t
I2C interface configuration.

Public Members

i2c_bus_handle_t **i2c_bus**
Handle of i2c bus

uint32_t **clk_speed**
I2C clock frequency for master mode, (no higher than 1MHz for now)

uint16_t **slave_addr**
I2C slave address

struct scr_interface_driver_t
Define common function for screen interface driver.

Public Members

scr_interface_type_t **type**
Interface bus type, see scr_interface_type_t struct

esp_err_t (***write_command**) (void *handle, **const** uint8_t *cmd, uint32_t length)
Function to write command

esp_err_t (***write_data**) (void *handle, uint16_t data)
Function to write a data

esp_err_t (***write**) (void *handle, **const** uint8_t *data, uint32_t length)
Function to write a block data

esp_err_t (***read**) (void *handle, uint8_t *data, uint32_t length)
Function to read a block data

esp_err_t (***bus_acquire**) (void *handle)
Function to acquire interface bus

esp_err_t (*bus_release) (void *handle)
Function to release interface bus

Enumerations

enum scr_interface_type_t
Type of screen interface.

Values:

SCREEN_IFACE_I2C
I2C interface

SCREEN_IFACE_8080
8080 parallel interface

SCREEN_IFACE_SPI
SPI interface

3.2 Digital Tube

[??]

Digital tube and dot matrix LEDs are common display solutions in embedded systems, which occupy fewer pins and memory resources than LCD displays and are simpler to implement, making them more suitable for application scenarios with single display requirements such as timing, counting, status display and etc.

The digital tube and LED display drivers adapted to ESP-IoT-Solution are shown in the following table:

Name	Features	Inter-face	Driver	Datasheet
CH450	Digital tube display driving chip, supports 6-bit digital tube	I2C	ch450	CH450
HT16C21	20×4/16×8 LCD controller, supports RAM mapping	I2C	ht16c21	HT16C21
IS31FL3XXX	Dot matrix LED controller	I2C	is31fl3xxx	IS31FL3XXX

3.2.1 CH450 Driver

CH450 is a digital tube display driving chip that can be used to drive a 6-bit digital tube or a 48-dot LED matrix and can communicate with ESP32 via the `I2C` interface.

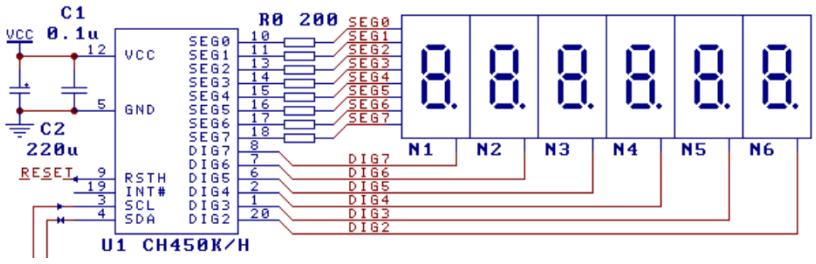


Fig. 5: CH450 Typical Application Circuit

This driver encapsulates the basic operations of CH450, and users can directly call `ch450_write()` or `ch450_write_num()` to display numbers on the digital tube.

Example

```
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);
```

3.2.2 HT16C21 Driver

HT16C21 is a LCD control/driver chip which supports RAM mapping and can be used to drive `20 x 4` or `16 x 8` segmented LCDs. The chip communicates with ESP32 via the `I2C` interface.

This driver encapsulates the basic operations of HT16C21. After creating an example using `ht16c21_create`, users can configure its parameters via `ht16c21_param_config` and then call `ht16c21_ram_write` directly to write data.

Example

```
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);
```

(continues on next page)

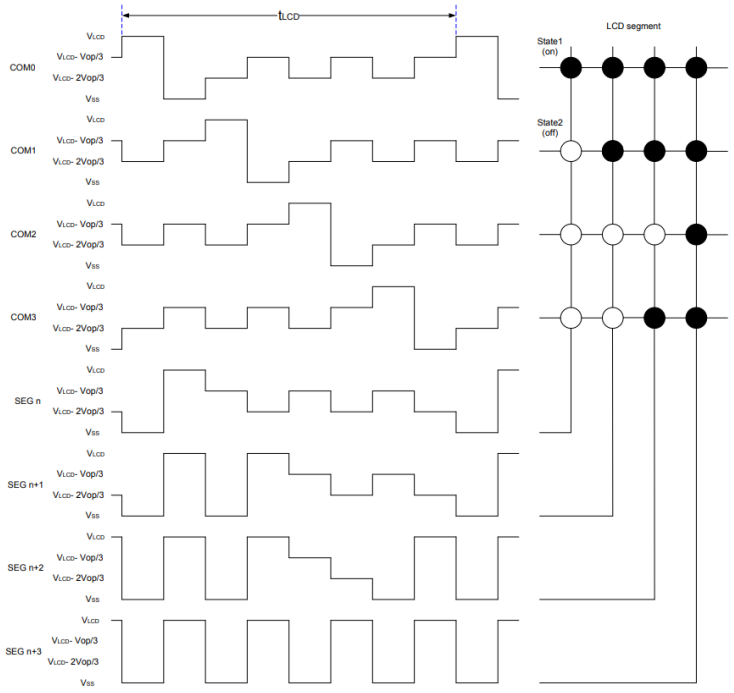


Fig. 6: HT16C21 Typical Drive Model

(continued from previous page)

```
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;
};
ht16c21_param_config(seg, &ht16c21_conf);
ht16c21_ram_write(seg, 0x00, lcd_data, 8);

ht16c21_delete(seg);
i2c_bus_delete(&i2c_bus);
```

3.2.3 IS31FL3XXX Driver

The IS31FL3XXX series chips can be used to drive dot matrix LED screens with different sizes. The IS31FL3218 supports 18 constant current channels, with each channel controlled by an independent PWM. It has a maximum output current of 38 mA and can directly drive LEDs for display. The IS31FL3736 supports more channels and can compose a maximum size of LED matrix as `12 x 8`. With each channel driven by an 8-bit PWM driver, the IS31FL3736 can support up to 256 gradients.

This driver encapsulates the basic operations of IS31FL3XXX. The example is shown in the next section.

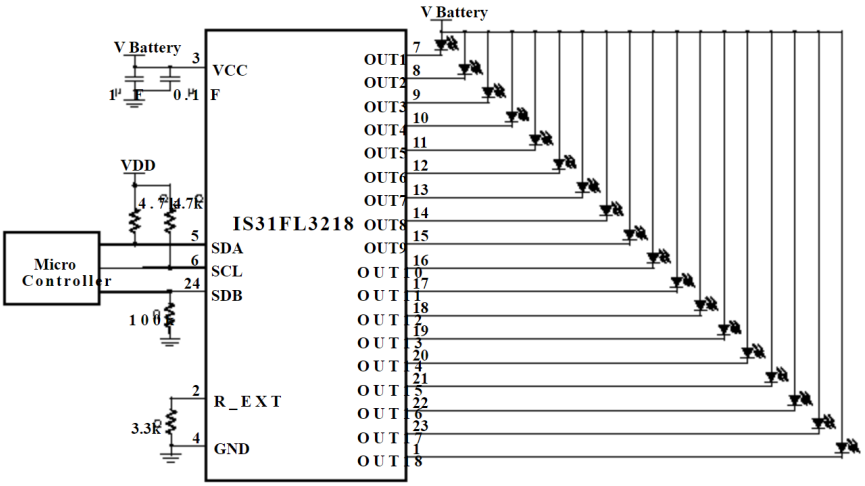


Fig. 7: IS31FL3218 Typical Application Circuit

IS31FL3218 Example

```
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);
```

3.3 LED Indicator

[??]

As one of the simplest output peripherals, LED indicators can indicate the current operating state of the system by blinking in different types. ESP-IoT-Solution provides an LED indicator component with the following features:

- Can define multiple groups of different blink types
- Can define the priority of blink types
- Can set up multiple indicators
- LEDC and other drivers support adjustable brightness, gradient

3.3.1 Instructions

Pre-define Blink Types

The blink step structure *blink_step_t* defines the type of a step, indicator state and the state duration. Multiple steps are combined into one blink type, and different blink types can be used to identify different system states. The blink type is defined as follows:

Example 1. define a blink loop: turn on 0.05 s, turn off 0.1 s, and loop.

```
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
};
```

Example 2. define a blink loop: turn on 0.05 s, turn off 0.1 s, turn on 0.15 s, turn off 0.1 s, and stop blink.

```
const blink_step_t test_blink_one_time[] = {
    {LED_BLINK_HOLD, LED_STATE_ON, 50},                 // step1: turn on LED 50 ms
    {LED_BLINK_HOLD, LED_STATE_OFF, 100},               // step2: turn off LED 100 ms
    {LED_BLINK_HOLD, LED_STATE_ON, 150},                // step3: turn on LED 150 ms
    {LED_BLINK_HOLD, LED_STATE_OFF, 100},               // step4: turn off LED 100 ms
    {LED_BLINK_STOP, 0, 0},                             // step5: stop blink (off)
};
```

Example 3. Define a cyclic blink: 0.05s fade, 0.5s fade, and the light goes off at the end of execution. (GPIO mode is not supported)

```
const blink_step_t test_blink_breathe[] = {
    {LED_BLINK_BREATHE, LED_STATE_ON, 500},             // step1: fade from off to␣
↪on 500ms
    {LED_BLINK_BREATHE, LED_STATE_OFF, 500},            // step2: fade from on to␣
↪off 500ms
    {LED_BLINK_BRIGHTNESS, 50, 500},                    // step3: set to half␣
↪brightness 500 ms
    {LED_BLINK_STOP, 0, 0},                             // step4: stop blink (50%␣
↪brightness)
};
```

After defining a blink type, you need to add its corresponding enumeration member to `led_indicator_blink_type_t` and then add the type to the blink type list `led_indicator_blink_lists`, as the following example:

```
typedef enum {
    BLINK_TEST_BLINK_ONE_TIME, /**< test_blink_one_time */
    BLINK_TEST_BLINK_LOOP,     /**< test_blink_loop */
    BLINK_MAX,                 /**< INVALIED 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,
};
```

Pre-define Blink Priorities

For the same indicator, a high-priority blink can interrupt an ongoing low-priority blink, which will resume execution after the high-priority blink stop. The blink priority can be adjusted by configuring the enumeration member order of the blink type `led_indicator_blink_type_t`, the smaller order value the higher execution priority.

For instance, in the following example, `test_blink_one_time` has higher priority than `test_blink_loop`, and should blink first:

```
typedef enum {
    BLINK_TEST_BLINK_ONE_TIME, /**< test_blink_one_time */
    BLINK_TEST_BLINK_LOOP,     /**< test_blink_loop */
    BLINK_MAX,                 /**< INVALIED type */
} led_indicator_blink_type_t;
```

Control Indicator Blinks

Create an indicator by specifying an IO and a set of configuration information.

```
led_indicator_config_t config = {
    .mode = LED_GPIO_MODE,
    .led_gpio_config = {
        .active_level = 1,
        .gpio_num = 1,
    },
    .blink_lists = led_indicator_get_sample_lists(),
    .blink_list_num = led_indicator_get_sample_lists_num(),
};
led_indicator_handle_t led_handle = led_indicator_create(8, &config); // attach to␣
↪gpio 8
```

Start/stop blinking: control your indicator to start/stop a specified type of blink by calling corresponding functions. The functions are returned immediately after calling, and the blink process is controlled by the internal timer. The same indicator can perform multiple blink types in turn based on their priorities.

```
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
```

Delete an indicator: you can also delete an indicator to release resources if there are no further operations required.

```
led_indicator_delete(&led_handle);
```

Preempt operation: You can flash the specified type directly at any time.

```
led_indicator_preempt_start(led_handle, BLINK_TEST_BLINK_LOOP);
```

Stop preempt: You can use the stop queueing function to cancel the blinking mode that is being queued.

```
led_indicator_preempt_stop(led_handle, BLINK_TEST_BLINK_LOOP);
```

Note: This component supports thread-safe operations. You can share the LED indicator handle `led_indicator_handle_t` with global variables, or use *led_indicator_get_handle* to get the handle in other threads via the LED's IO number for operation.

Custom light blink

```
static blink_step_t const *led_blink_lst[] = {
    [BLINK_DOUBLE] = double_blink,
    [BLINK_TRIPLE] = triple_blink,
    [BLINK_NUM] = NULL,
};

led_indicator_config_t config = {
    .mode = LED_GPIO_MODE,
    .led_gpio_config = {
        .active_level = 1,
        .gpio_num = 1,
    },
    .blink_lists = led_blink_lst,
    .blink_list_num = BLINK_MAX,
};
```

By defining `led_blink_lst[]` to achieve the custom indicator.

Adjustment of Gamma

The way human eyes perceive brightness is not linear but has certain nonlinear characteristics. Under normal conditions, the human eye is more sensitive to darker areas and less sensitive to brighter areas. However, on digital display devices such as monitors, the brightness values of images are usually encoded in a linear manner. This leads to issues of brightness distortion or loss of details when converting the linearly encoded brightness values to the perceived brightness by the human eye. To address this problem, gamma correction is applied to the image. Gamma correction involves adjusting the brightness values nonlinearly to correct the image display. By applying a gamma value (typically ranging from 2.2 to 2.4), the linearly encoded brightness values are mapped to a nonlinear brightness curve that better matches the perception of the human eye. This improves the visibility of details in darker areas and enhances the overall visual accuracy and balance of the image.

```
float gamma = 2.3;
led_indicator_new_gamma_table(gamma);
```

The default gamma table is 2.3, and a new gamma table can be generated using the *led_indicator_new_gamma_table()* function.

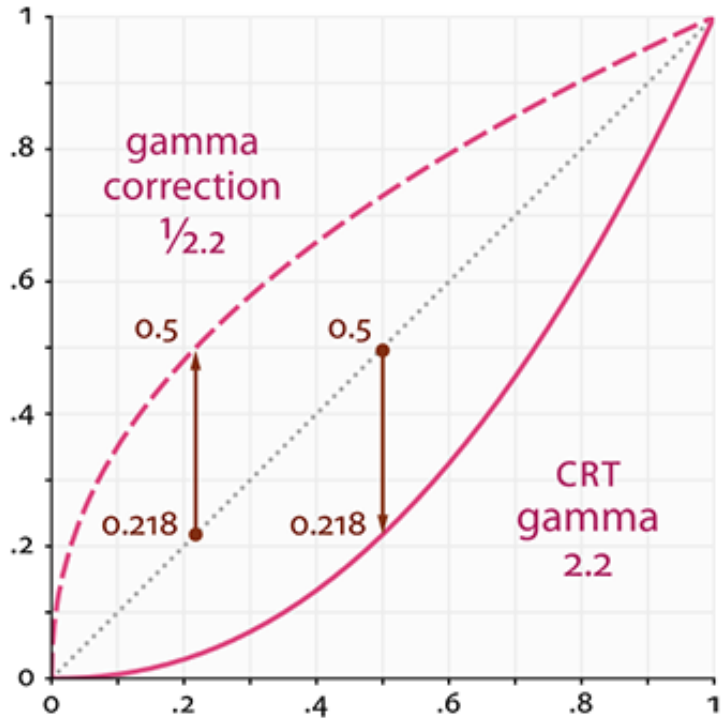


Fig. 8: Gamma Curve

3.3.2 API Reference

3.3.3 Header File

- led/led_indicator/include/led_indicator.h

3.3.4 Functions

led_indicator_handle_t **led_indicator_create**(**const** *led_indicator_config_t* **config*)
create a LED indicator instance with GPIO number and configuration

Return led_indicator_handle_t handle of the LED indicator, NULL if create failed.

Parameters

- `config`: configuration of the LED, eg. GPIO level when LED off

led_indicator_handle_t **led_indicator_get_handle**(void **hardware_data*)
get the handle of created led_indicator with hardware data

Return led_indicator_handle_t handle of the created LED indicator, NULL if not created.

Parameters

- `hardware_data`: user hardware data for LED

esp_err_t **led_indicator_delete**(*led_indicator_handle_t handle*)
delete the LED indicator and release resource

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK Success
- ESP_FAIL Delete fail

Parameters

- `handle`: pointer to LED indicator handle

esp_err_t **led_indicator_start**(*led_indicator_handle_t handle*, int *blink_type*)

start a new blink_type on the LED indicator. if mutiple blink_type started simultaneously, it will be executed according to priority.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_FOUND no predefined blink_type found
- ESP_OK Success

Parameters

- `handle`: LED indicator handle
- `blink_type`: predefined blink type

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.

Return esp_err_t

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_FOUND no predefined blink_type found
- ESP_OK Success

Parameters

- `handle`: LED indicator handle
- `blink_type`: predefined blink type

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().

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_INVALID_ARG if parameter is invalid

Parameters

- `handle`: LED indicator handle
- `blink_type`: predefined blink type

esp_err_t **led_indicator_preempt_stop**(*led_indicator_handle_t* *handle*, int *blink_type*)
Stop the current preemptive action.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_INVALID_ARG if parameter is invalid

Parameters

- `handle`: LED indicator handle
- `blink_type`: predefined blink type

uint8_t **led_indicator_get_current_fade_value**(*led_indicator_handle_t* *handle*)
Get the current fade value of the LED indicator.

Return uint8_t Current fade value: 0-255 if handle is null return 0

Parameters

- `handle`: LED indicator handle

3.3.5 Structures

struct blink_step_t
one blink step, a meaningful signal consists of a group of steps

Public Members

blink_step_type_t **type**
action type in this step

uint8_t **value**
hold on or off, set NULL if LED_BLINK_STOP or LED_BLINK_LOOP

uint32_t **hold_time_ms**
hold time(ms), set NULL if not LED_BLINK_HOLD,

struct led_indicator_config_t
LED indicator specified configurations, as a arg when create a new indicator.

Public Members

led_indicator_mode_t **mode**
LED work mode, eg. GPIO or pwm mode

led_indicator_gpio_config_t ***led_indicator_gpio_config**
LED GPIO configuration

led_indicator_ledc_config_t ***led_indicator_ledc_config**
LED LEDC configuration

led_indicator_custom_config_t ***led_indicator_custom_config**
LED custom configuration

union *led_indicator_config_t*::**[anonymous] [anonymous]**
LED configuration

blink_step_t **const** ****blink_lists**
user defined LED blink lists

uint16_t **blink_list_num**
number of blink lists

3.3.6 Type Definitions

typedef void ***led_indicator_handle_t**
LED indicator operation handle

3.3.7 Enumerations

enum [anonymous]
LED state: 0-100, only hardware that supports to set brightness can adjust brightness.

Values:

LED_STATE_OFF = 0
turn off the LED

LED_STATE_25_PERCENT = 64
25% brightness, must support to set brightness

LED_STATE_50_PERCENT = 128
50% brightness, must support to set brightness

LED_STATE_75_PERCENT = 191
75% brightness, must support to set brightness

LED_STATE_ON = UINT8_MAX
turn on the LED

enum blink_step_type_t
actions in this type

Values:

LED_BLINK_STOP = -1
stop the blink

LED_BLINK_HOLD
hold the on-off state

LED_BLINK_BREATHE
breathe state

LED_BLINK_BRIGHTNESS
set the brightness

LED_BLINK_LOOP
loop from first step

enum led_indicator_mode_t
LED indicator blink mode, as a member of *led_indicator_config_t*.

Values:

LED_GPIO_MODE
blink with max brightness

LED_LEDC_MODE
blink with LEDC driver

LED_CUSTOM_MODE
blink with custom driver

CHAPTER

FOUR

USB HOST & DEVICE

[??]

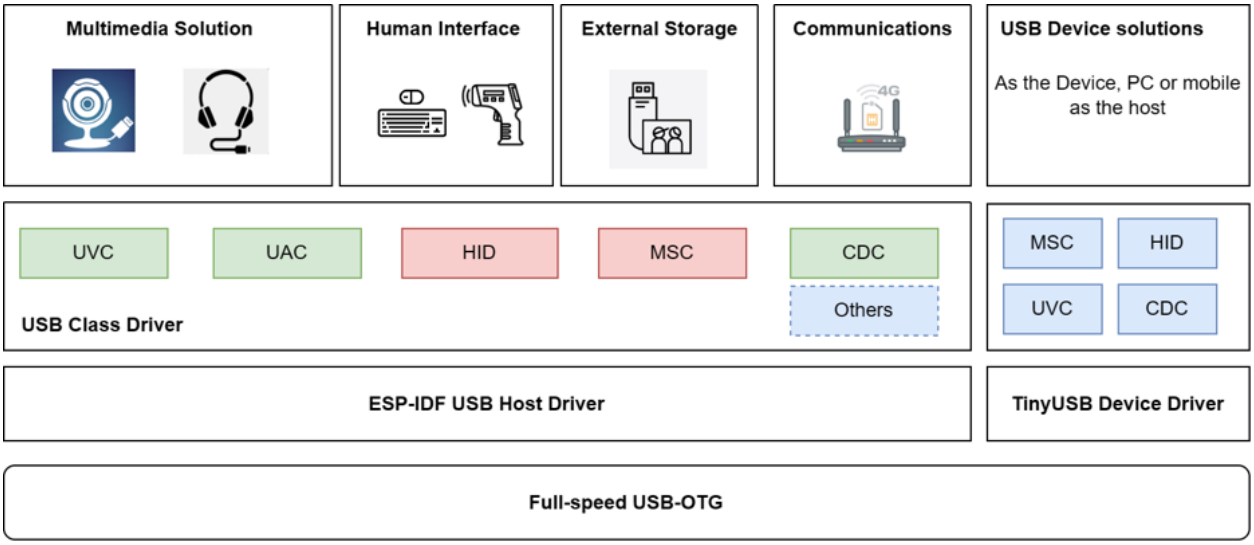
4.1 ESP USB ????

4.1.1 USB ??

USB?Universal Serial Bus??????????????????????????USB ??????? USB ??? USB ??????????????????????

USB IF?USB Implementers Forum?? USB ?????????? USB ????? USB 1.1?USB 2.0?USB 3.0 ????? USB ?????????????????????????????? USB ????Device Class????????????? HID?Human Interface Device?????????MSC?Mass Storage Class????????????CDC?Communication Device Class????????Audio?Video ??

?? ESP32-S2/S3/C3 ??????? USB-OTG ? USB-Serial-JTAG ????????? USB ????? USB ???????USB ???????USB ???????USB ?????????



4.1.2 USB ????

Type-A ??? USB ????????

Pin	Name	Cable color	Description
1	VBUS	**Red**	+5V
2	D-	**White**	Data-?0?3.3V?
3	D+	**Green**	Data+?0?3.3V?
4	GND	**Black**	Ground

- ?????????????? 1 ??? IO ?? VBUS ????????????????
- D- D+ ??????????????????????

4.1.3 USB-OTG Full-speed ?????

USB OTG Full-speed ??????????? USB-OTG?USB Host ? USB Device ?????????????????????? Full-speed (12Mbps) ? Low-speed (1.5Mbps) ???????? USB 1.1 ? USB 2.0 ???

ESP-IDF ? v4.4 ??????? USB Host ? USB Device ??????????????????????

?????????USB-OTG ???????

4.1.4 USB-Serial-JTAG ?????

USB-Serial-JTAG Controller????? USB Serial ? USB JTAG ????? USB ????????? USB ?????????? log?CDC ??? JTAG ????????? USB ????????????????

?????????USB-Serial-JTAG ???????

4.1.5 USB Full-speed PHY ??

USB Full-speed PHY??? USB Full-speed Transceiver??? USB Controller ????? USB ?????????????????????? USB Full-speed PHY ???????? IO ???

?????????USB-PHY ???

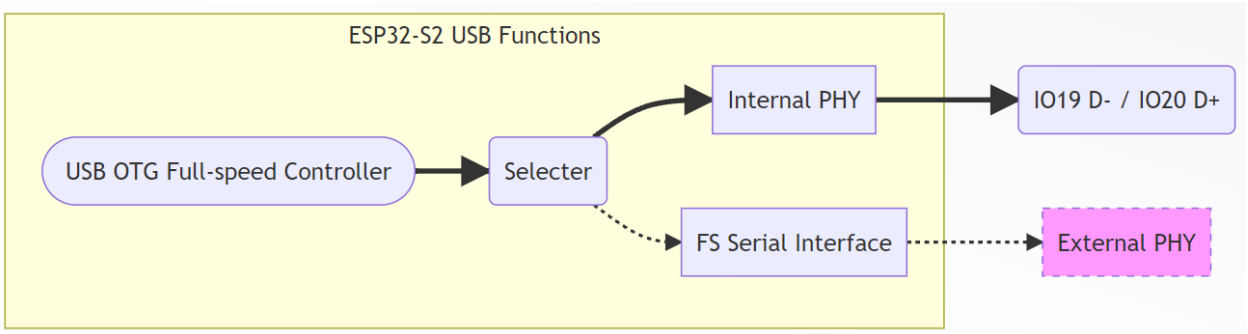
4.1.6 ESP32-S/C ?? USB ??????

	USB OTG High-speed	USB OTG Full-speed	USB-Serial-JTAG	Fulls-peed 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

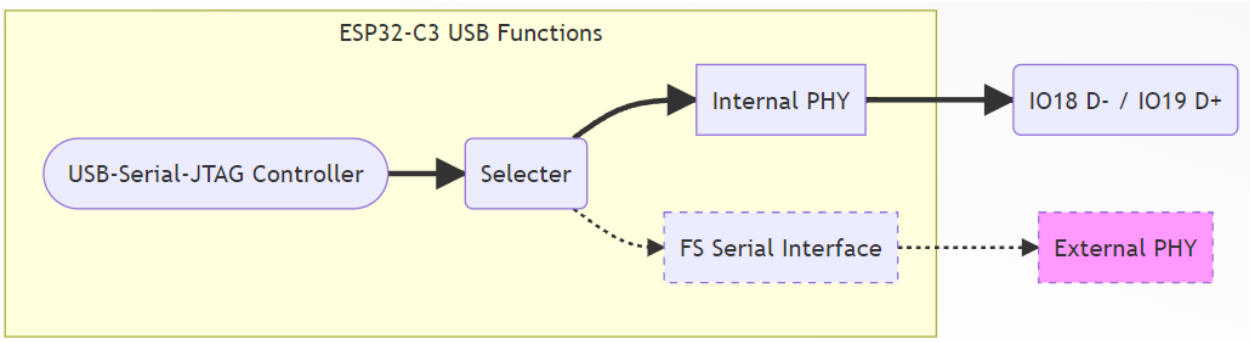
4.1.7 ESP32-S2 USB ????

ESP32-S2 ?? **USB OTG Full-speed Controller ? USB Full-speed PHY**?????????



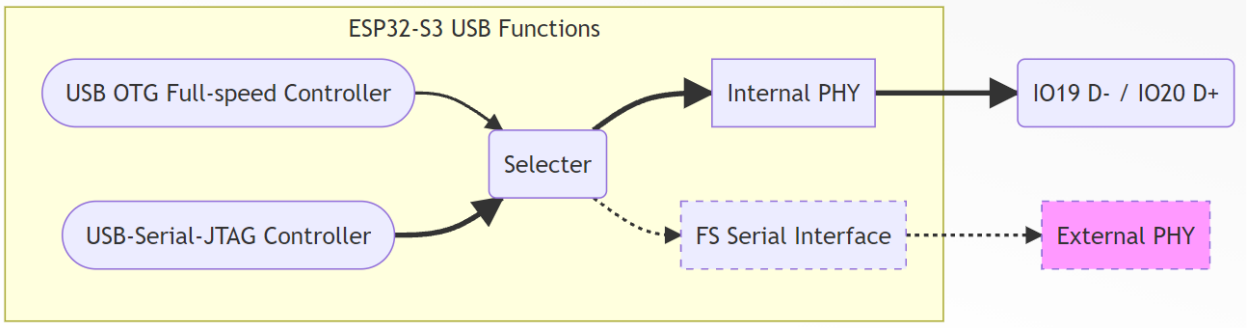
4.1.8 ESP32-C3 USB ????

ESP32-C3 ?? **USB-Serial-JTAG Controller** ? **USB Full-speed PHY**????????:



4.1.9 ESP32-S3 USB ????

ESP32-S3 ???? USB ???????? **USB OTG Full-speed Controller** ? **USB-Serial-JTAG Controller**????? **USB Full-speed PHY**??? USB PHY ????? **USB-Serial-JTAG** ????????? eFuse ?????????????????????????????? PHY????????????????? USB PHY ????????? USB PHY ???



4.2 USB-OTG ????

ESP32-S2/S3 ?????? USB-OTG ???????? USB ????? USB PHY?????? USB ????? PC???? USB Host ? USB Device ???

4.3 USB-OTG ????

ESP32-S2/S3 USB-OTG Full Speed ???????? 12 Mbps????? USB ?????????????????????????????? 12 Mbps??????????????????????

?????	??	??	??	??
?????	?????????	?????	?????????	????????
?????	?	?	?	?
?????	?	?	?	?
???????	?	?	?	?
???????	??10%?	??90%?	?	??90%?
???????	?	?	?	?
????????	64??	64??	64??	~512??
?????????	•	1	19	1
???????	•	64000 Bytes/s	1216000 Bytes/s	512000 Bytes/s

- ????????????????? (Bytes/s) = ???????? * ????????? * 1000
- ??

4.4 USB-OTG ??????

4.4.1 ?? USB OTG Console ???????? LOG

ESP32-S2/S3 ??? USB-OTG ???????ROM Code ???? USB ????? (CDC) ?????????????? UART ????? Log?Console ?????????

1. ?? USB OTG Console ??

    1. ? `menuconfig` ???? USB OTG Console ??????????

    2. ????? Boot ????????????????? USB ???? PC?????????PC ????????????Windows ? `COM*`?Linux ? `/dev/ttyACM*`, MacOS ? `/dev/cu*`?

    3. ?? esptool ????????? idf.py flash???????????????????

2. ??????????USB OTG Console ???????????????? USB ???? PC?PC ????????????Windows ? `COM*`?Linux ? `/dev/ttyACM*`, MacOS ? `/dev/cu*`?LOG ????????????????

3. ?????????? Boot ???????? esptool ????????? idf.py flash????????????????????????????????esptool ?? USB ?????????? Reset ??????????

    ???????????????USB OTG Console

4.4.2 ?? USB OTG DFU ?????

ESP32-S2/S3 ??? USB-OTG ???????ROM Code ???? USB DFU?Device Firmware Upgrade????????????? DFU ??????

1. ?? DFU ????????????????????????????? Boot ???????????? USB ???? PC?

2. ?????????? `idf.py dfu` ?? DFU ???????? `idf.py dfu-flash` ??????

3. ??????? DFU ?????????? `idf.py dfu-list` ?? DFU ????????? `idf.py dfu-flash --path <path>` ????????

    ???????????????Device Firmware Upgrade via USB

4.5 ?? USB-OTG ???? USB Host ??

USB-OTG ???? USB Host ????????? USB ????????? USB ???ESP-IDF ? v4.4 ????????? USB Host Driver??????? ESP-IDF USB Host??? USB Class Driver?

???????????? USB Host HID?USB Host MSC?USB Host CDC?USB Host UVC ??????????????????????????????

USB Host ????????? USB Host Solution?

4.6 ?? USB-OTG ???? USB Device ??

USB-OTG ???? USB Device ???????????? TinyUSB ???????????? TinyUSB ????????? USB ???????????? HID?MSC?CDC?ECM, UAC ??

USB Device ????????? USB Device Solution?

4.7 USB-Serial-JTAG ????

ESP32-S3/C3 ????? USB-Serial-JTAG ??????? USB-to-serial ???? USB-to-JTAG ????????? USB ???? PC????????????????? LOG ????USB-Serial-JTAG ????????? ESP32-C3 ??????-USB Serial/JTAG Controller?

4.8 USB-Serial-JTAG ????

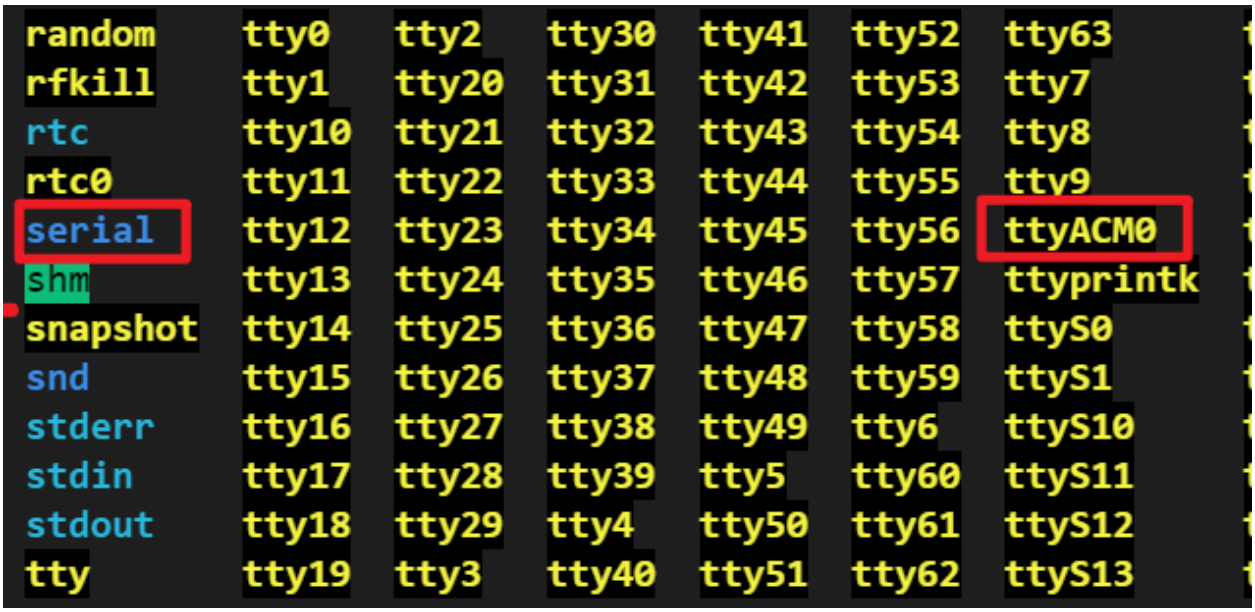
- Linux ? MacOS ???????????
- Windows 10 ????????????????
- Windows 7/8 ??????????????????esp32-usb-jtag-2021-07-15????????? ESP-IDF Windows Installer??? USB-Serial-JTAG ????????

4.9 USB-Serial-JTAG ??????

USB-Serial-JTAG ???? PC ????????????????

Windows ???????

Linux ??????



4.9.1 ?? USB-Serial-JTAG ????

- ???????USB-Serial-JTAG ???????????????????? USB ???? PC?????? esptool ????????? idf.py flash??? USB-Serial-JTAG ?????????Windows ? `COM*`?Linux ? `/dev/ttyACM*`, MacOS ? `/dev/cu*`???????????esptool ?? USB ????????? Reset ?????????
- ????????? USB-Serial-JTAG ??? USB ???????????????? GPIO ????? IO?USB-Serial-JTAG ???? USB ???????????? USB ???????????????? Boot ?????????????????????????? esptool ?????
- ???????????? USB-Serial-JTAG ??? USB ???????????????? USB ?????????????????????????? Boot ?????
- ????????? USB ????????????COM ?????????????????????? ?? Windows ?? USB ???????? COM ???

4.9.2 ?? USB-Serial-JTAG ????

USB-Serial-JTAG ???? JTAG ?????????????? USB ???? PC????? OpenOCD ?????????????????? ?? ESP32-C3 ?? JTAG ???

4.9.3 ?? USB-Serial-JTAG ???? LOG

- ????? menuconfig-> Component config → ESP System Settings → Channel for console secondary output ?? USB-Serial-JTAG LOG ?????????
- LOG ?????????????? USB ???? PC????? idf.py monitor ????????? USB-Serial-JTAG ?????????Windows ? COM*?Linux ? /dev/ttyACM*, MacOS ? /dev/cu*????????? LOG?
- USB-Serial-JTAG ?????????? LOG?????????USB-Serial-JTAG ?????????? LOG?
- USB-Serial-JTAG LOG ????????????????? deep sleep ? light sleep ????????????????? LOG????? UART ???

4.9.4 ?? USB-Serial-JTAG ????? GPIO

????????????? USB-Serial-JTAG ??? USB ????????????? GPIO????? USB D+ ?????? USB ????????? USB D+ ????????????? GPIO ??????

- ESP-IDF v4.4 ????? GPIO ??????? USB D+ ????????? GPIO ???????????
- ????????????? USB_SERIAL_JTAG.conf0.dp_pullup = 0; ? USB D+ ????????

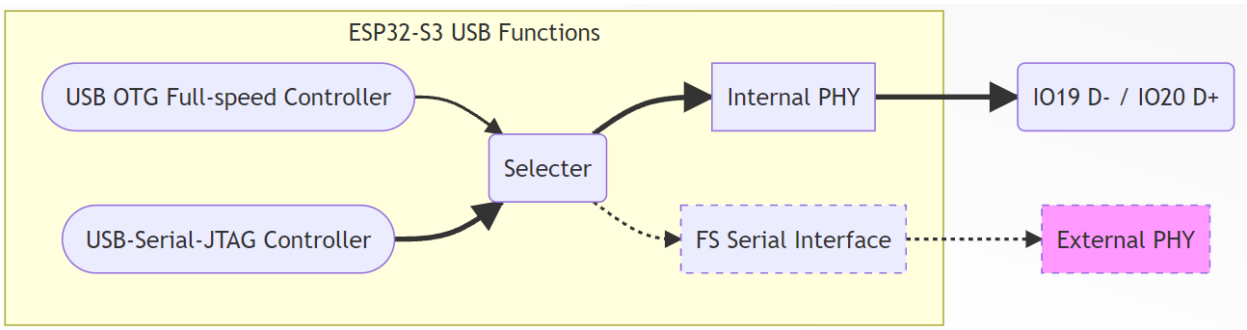
????????? USB D+ ????????????????????????????????USB D+ ????????? D+ ???? GPIO ????????????????????? USB D+ ????????????????????? USB D+ ???????????

4.10 USB PHY/Transceiver ??

USB Full-speed PHY/Transceiver ????? USB ?????????? USB ????????????????????????ESP32-S2/S3 ????????? USB Full-speed PHY????????????? USB D+ D- ??? USB ?????????ESP32-S2/S3 ??????? PHY ???????????????????? PHY?

4.11 ???? PHY

ESP32-S2/S3/C3 ????? USB PHY???????? PHY ?????????? USB ??????? USB D+/D- ?????????? USB ????????? ESP32-S3 ?? USB-OTG ? USB-Serial-JTAG????????? PHY????????????????



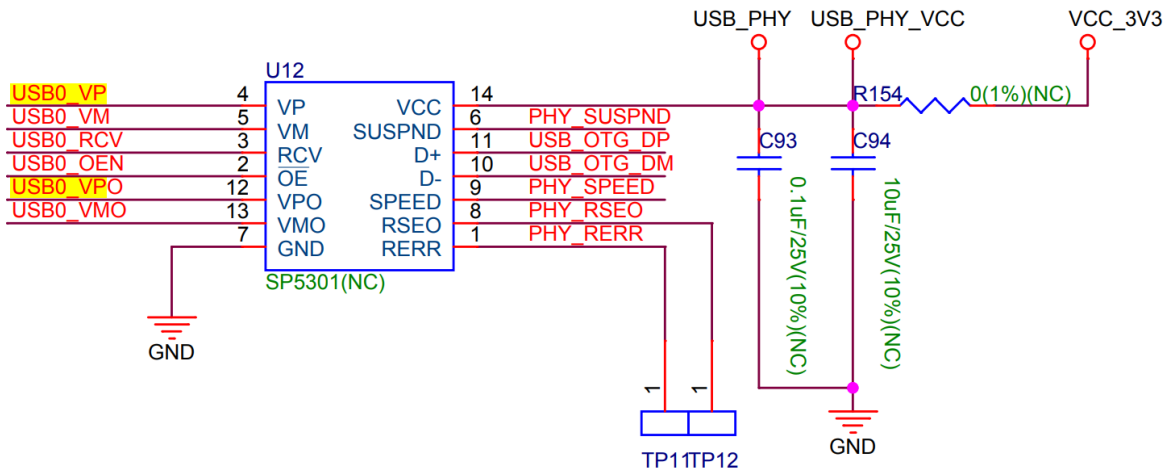
□□ USB-PHY □□□□□ GPIO□□□□□□□

	D+	D-
ESP32-S2	20	19
ESP32-S3	20	19
ESP32-C3	19	18
ESP32-C6	13	12

4.12 □□□□ PHY

□□□□□□□□ PHY□□□□□ USB-OTG □ USB-Serial-JTAG □□□□□□□□□

ESP32S2/S3 □□ SP5301 □□□□□□ USB PHY□□□ PHY □□□□□□□□□



□□□□ USB PHY□□□□□□ 6 □ GPIO

4.13 USB PHY □□□□

1. □□□□□□ USB-OTG □ USB-Serial-JTAG □□□□□□□□□□□□□ USB-Serial-JTAG □□□ USB-PHY □□□□□□□□□□□ USB □□□□□□□□□□□□□□□□□□□□
2. □□□□ USB Host Driver □ TinyUSB □□□□□ USB-OTG □□□□□□□□□□□□USB-PHY □□□□□□□□ USB-OTG□□□□□□□□□□□□□□ USB-OTG □□□□□□□□□□□□□ USB-Serial-JTAG □□□□□□□□□□□ Boot □□□□□□□

4.14 ?? USB PHY ????

?? 1? ????????? USB-PHY ????? USB-OTG?

- USB Host Driver ? TinyUSB ????????? USB PHY ??????? USB-PHY ????? USB-OTG????????????????? USB PHY ?? API?

?? 2?????? efuse usb_phy_sel ?? 1?? USB-PHY ??????? USB-OTG?

- ??????? Boot ????? USB-OTG ????????? efuse ???????????? Boot ??????? USB-OTG ????????? USB DFU?
- ????? efuse ? 1 ?????? 0?? USB-PHY ??????? USB-OTG????? Boot ??????? USB-OTG ???USB-Serial-JTAG ?????????
- ?????? DateCode 2219 ??? ESP32-S3 ?????? (PW No. ?? PW-2022-06-XXXX), ?? EFUSE_DIS_USB_OTG_DOWNLOAD_MODE (BLK0 B19[7]) ?????? 1 ?USB OTG ???????????????? efuse_usb_phy_sel ?? 1????????? Boot ????USB-Serial-JTAG ? USB-OTG ???????????

4.15 USB VID ? PID

VID ? PID ? USB ???????????????????? USB ????? VID ? PID ? USB-IF ???USB-IF ? USB ??????????

4.15.1 ?????????? VID ? PID

- ????????? USB Host ????????? VID ? PID?
- ????????? USB Device ????????? VID?0x303A?, ???? TinyUSB ????? USB ???????????? PID??? TinyUSB ?? PID ???

4.15.2 ?? VID ? PID

???????????? USB ???????????????? VID (Vendor ID) ? PID (Product ID)?

- ????????? USB-IF ??? VID????? ???? USB-IF ??????? USB-IF ????? VID ? PID?
- ???????????? VID????????? PID????, ??????? ???? PID?

????????? VID ? TinyUSB ??? PID, ????????????? USB ????????? USB ???

4.16 USB ??

USB ???? USB Implementers Forum?USB-IF????????????????? USB ??????????????????????

USB ??????????????????????????????

- ????????? USB ??????? USB ???
- ????????? USB ????????????? USB ???

????????????? https://www.usb.org ??? USB-IF ????????

4.17 USB Host ??

ESP32-S2/S3 ????? USB-OTG ???????? USB ????????? USB ????????? ESP32-S2/S3 ????? USB Host ??????

4.17.1 ESP USB Camera ????

???? USB ????????????? MJPEG ???????????????? 480*800@15fps??????????????????????????????

??:

- ????
- ??????
- ?? UVC1.1/1.5 ???????
- ??????????
- ??????????
- ?? MJPEG ??????
- ???????????????

??:

- ??? ESP32-S2?ESP32-S3
- ???USB-OTG
- USB ??????? MJPEG ????????? 800*480@15fps????????? 480*320@15fps????????? usb_stream API ??

??:

- usb_stream ??
- usb_stream API ??
- USB Camera Demo ??
- ????: USB ??? + WiFi ?? usb/host/usb_camera_mic_spk
- ????: USB ??? + LCD ???? usb/host/usb_camera_lcd_display

4.17.2 ESP USB Audio ????

???? USB ???? USB ???????? PCM ???????????????????? 48KHz 16bit ??????? 48KHz 16bit ??????? Type-C ?????????????????????????? UVC ??????????????????

??:

- ????
- ??????
- ??????????
- ?? PCM ??????
- ??????????
- ??????????
- ??????????
- ??????????
- ??? USB Camera ????

??:

- ??? ESP32-S2?ESP32-S3
- ???USB-OTG
- USB ???????? PCM ??

??:

- usb_stream ??
- usb_stream API ??
- USB Audio Demo ??
- ????: MP3 ?????? + USB ?? usb/host/usb_audio_player

4.17.3 ESP USB 4G ????

???? USB ???? 4G Cat.1?Cat.4 ?????? PPP ?????????? Wi-Fi SoftAP ????????????????????????????MiFi ??????????????????????

??:

- ????
- ?????
- ?? Modem+AT ??????????
- ?? PPP ???? ???? 4G ??????
- ?? 4G ? Wi-Fi ??
- ?? NAPT ??????
- ??????
- ?????????
- ??????????
- ?????????

??:

- ??? ESP32-S2?ESP32-S3
- ???USB-OTG
- 4G ????? Cat.1 Cat.4 ????? 4G ????????? PPP ??

??:

- USB 4G Demo ??
- iot_usbh_modem ??
- ????: 4G Wi-Fi ??? usb/host/usb_cdc_4g_module

4.17.4 ESP USB ????

???? USB ?????? U ?????? USB3.1/3.0/2.0 ?? U ?????? U ???? FatFS ??

??:

- ?? USB3.1/3.0/2.0 U ?
- ?????? 32G
- ?????
- ?? Fat32/exFat ??
- ?????????
- ?? U ? OTA

??:

- ??? ESP32-S2?ESP32-S3
- ???USB-OTG
- U ??????? Fat32 ???????? 32GB ?? U ????? 32GB ?????????? exFat

??:

- usb_host_msc ??
- ?????: ?? U ? + ???????

4.18 USB Device ??

4.19 ??? USB ??????

?? USB ?????USB ???????????? 5V VBUS ?? VBUS ??????????????????????????

USB ?? VBUS ????????????????? USB PHY ????????? **ADC/GPIO ??????**

?? ESP32S2/S3 ?? USB PHY ?????????????????????? ADC/GPIO ?????????? GPIO ??????????????????

?? ESP-IDF 4.4 ?????:

1. ???????????? IO??????????????????????????????????????? 100KΩ ?? ESP32S2/S3 ?? (ESP32S2/S3 IO ????????? 3.3v)?
2. ? `tinyusb_driver_install` ???????? `usbd_vbus_detect_gpio_enable` ???? VBUS ????????????????????????????????

```
/**
 *
 * @brief For USB Self-power device, the VBUS voltage must be monitored to achieve␣
↪hot-plug,
 *        The simplest solution is detecting GPIO level as voltage signal.
 *        A divider resistance Must be used due to ESP32S2/S3 has no 5V tolerate pin.
 *
 *    5V VBUS ?????????     ?????????    GND
 *     ????????? 100K ??????? 100K ???????
 *             ????????? ?  ?????????
 *                      ?           GPIOX
 *                      ???????????????????
 *        The API Must be Called after tinyusb_driver_install to overwrite the␣
↪default config.
 * @param gpio_num, The gpio number used for vbus detect
 *
 */
static void usbd_vbus_detect_gpio_enable(int gpio_num)
{
    gpio_config_t io_conf = {
        .intr_type = GPIO_INTR_DISABLE,
        .pin_bit_mask = (1ULL<<gpio_num),
        //set as input mode
```

(continues on next page)

(continued from previous page)

```
        .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_SESSEND_IN_IDX, 1);
    return;
}
```

?? ESP-IDF 5.0 ?????:

1. ????????????? IO????????????????????????????? 100KΩ ?? ESP32S2/S3 ???
2. ????? VBUS ? IO ???? GPIO ????;
3. ??? IO ??? `tinyusb_config_t` ????????

```
#define VBUS_MONITORING_GPIO_NUM GPIO_NUM_4
// Configure GPIO Pin for vbus monitoring
const gpio_config_t vbus_gpio_config = {
    .pin_bit_mask = BIT64(VBUS_MONITORING_GPIO_NUM),
    .mode = GPIO_MODE_INPUT,
    .intr_type = GPIO_INTR_DISABLE,
    .pull_up_en = false,
    .pull_down_en = false,
};
ESP_ERROR_CHECK(gpio_config(&vbus_gpio_config));
const tinyusb_config_t tusb_cfg = {
    .device_descriptor = &descriptor_config,
    .string_descriptor = string_desc_arr,
    .string_descriptor_count = sizeof(string_desc_arr) / sizeof(string_desc_arr[0]),
    .external_phy = false,
    .configuration_descriptor = desc_configuration,
    .self_powered = true,
    .vbus_monitor_io = VBUS_MONITORING_GPIO_NUM,
};
ESP_ERROR_CHECK(tinyusb_driver_install(&tusb_cfg));
```

4.20 ?? Windows ?? USB ???????? COM ??

???????? Windows PC ???????? VID?PID ? Serial ????????? 3 ?????????????????? PC ????????????????????????? COM ????????? Windows USB device registry entries?

ESP ROM Code ?? USB ?????????

	ESP32S2	ESP32S3	ESP32C3
VID	0x303a	0x303a	0x303a
PID	0x0002	0x1001	0x1001
Serial	0	MAC ?????	MAC ?????

- ESP32S2?usb-otg?Serial ??? 0?????????COM ???
- ESP32S3?usb-serial-jtag??ESP32C3?usb-serial-jtag?Serial ??? MAC ?????????????COM ?????

□□ COM □□□□□□□□□□□□□□□□□□□□□□□□ USB □□□□□□□□□□□□□□□ Windows □□□ COM □□□□□□□□□□□ Serial □□□□□□□

4.21 □□□□

□□□□□□□ `Windows CMD` , □□□□□□□□□□□□□□□□□□□□□□□□□□ Serial □□□□□□□□□□□□□□□□□□□□□□□□

```
REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags\303A10010101 /V␣
↪IgnoreHWSerNum /t REG_BINARY /d 01
```

□□□□□□ `ignore_hwserial_esp32s3c3.bat` □□□□□□□□□□□□□□□□

4.22 USB Stream Component

[□□]

`usb_stream` is an USB `UVC` + `UAC` host driver for ESP32-S2/ESP32-S3, which supports read/write/control multimedia streaming from usb device. For example, at most one UVC + one Microphone + one Speaker streaming can be supported at the same time.

Features:

1. Support video stream through UVC Stream interface, support both isochronous and bulk mode
2. Support microphone stream and speaker stream through the UAC Stream interface
3. Support volume, mute and other features control through the UAC Control interface
4. Supports automatic parse of device configuration descriptors
5. Support stream separately suspend and resume

4.22.1 USB Stream User Guide

- Development board
  1. Any `ESP32-S2` / `ESP32-S3` board with USB Host port can be used. Note that the port must be able to output voltage.
- USB UVC function
  1. Camera must be compatible with `USB1.1` full-speed mode
  2. Camera must support `MJPEG` output
  3. Users can manually specify the camera interface, transmission mode, and image frame params through *`uvc_streaming_config()`*
  4. For isochronous mode camera, interface `max packet size` should no more than `512` bytes
  5. For isochronous mode camera, frame stream bandwidth should be less than `4 Mbps` (500 KB/s)
  6. For bulk mode camera, frame stream bandwidth should be less than `8.8 Mbps` (1100 KB/s)
  7. Please refer example readme for special camera requirements
- USB UAC function
  1. Audio function must be compatible with `UAC1.0` protocol

2. Users need to manually specify the spk/mic sampling rate, bit width params through *uac_streaming_config()* function

- USB UVC + UAC

1. The UVC and UAC functions can be enabled separately. For example, only the UAC be configured to drive a usb headset, or only the UVC be configured to drive a USB camera
2. If you need to enable UVC and UAC at the same time, note that the driver currently only supports `Composite devices` with both camera and audio interfaces, rather than two separate devices.

4.22.2 USB Stream Config Reference

1. Using *uvc_config_t* to configure camera resolution and frame rate parameters. Using *uac_config_t* to configure the audio sampling rate, bit width and other parameters. The parameters are described as follows:

```
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 frequence, Hz
    .spk_bit_resolution = 16, //speaker resolution, bits
    .spk_samples_frequence = 16000, //speaker frequence, 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. Use the *uvc_streaming_config()* to config the UVC driver, or use the *uac_streaming_config()* to config the UAC driver
3. Use the *usb_streaming_start()* to turn on the stream, then the driver will handle USB connection and negotiation.
4. The host will matches the descriptors of the connected devices according to the user parameters. If the device fails to meet the configuration requirements, the driver prompt warning message
5. If the device meets user configuration requirements, the Host will continue to receive the IN stream (UVC and UAC mic), and will call the user's callbacks when new frames ready.
    1. The camera callback will be triggered after a new MJPEG image ready. The callback can block during processing, because which works in an independent task context.

2. The mic callback will be triggered after `mic_min_bytes()` bytes data received. But the callback here must not block in any way, otherwise it will affect the reception of the next frame. If the block operations for mic is necessary, you can use the polling mode instead of the callback mode through *`uac_mic_streaming_read()`* api.

6. User can send speaker OUT stream using *`uac_spk_streaming_write()`* through a ringbuffer, the Host will fetch the data when USB is free to send.
7. Use the *`usb_streaming_control()`* to control the stream suspend/resume, uac volume/mute control can also be support if the USB device has such feature unit;
8. Use the *`usb_streaming_stop()`* to stop the usb stream, USB resource will be completely released.

4.22.3 Bug report

ESP32-S2 ECO0 Chip SPI screen jitter when work with usb camera

1. In earlier versions of the ESP32-S2 chip, USB transfers can cause SPI data contamination (esp32s2>=ECO1 and esp32s3 do not have this bug)
2. Software workaround

- `spi_ll.h` add below function

```
//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;
}
```

- modify `spi_new_trans` implement as below

```
// 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);
#if 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);
}
```

4.22.4 Examples

usb/host/usb_camera_mic_spk

4.22.5 API Reference

Header File

- 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.

Return 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

Parameters

- `config`: parameters defined in uvc_config_t

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.

Return 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

Parameters

- `config`: parameters defined in uvc_config_t

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.

Return 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 resourse will be free.

Return 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.

Return esp_err_t ESP_ERR_INVALID_STATE: usb streaming not started ESP_ERR_TIMEOUT: timeout ESP_OK: device connected

Parameters

- `timeout_ms`: timeout in ms

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.

Return esp_err_t

- ESP_OK Success
- ESP_ERR_INVALID_STATE USB streaming is running, callback need register before start

Parameters

- `cb`: A pointer to a function that will be called when the USB streaming state changes.
- `user_ptr`: user_ptr is a void pointer.

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.

Return 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

Parameters

- `stream`: stream type defined in usb_stream_t
- `ctrl_type`: control type defined in stream_ctrl_t
- `ctrl_value`: control value

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.

Return 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

Parameters

- `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.

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.

Return 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

Parameters

- `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.

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 frequence. IF list_size equals 1 and the samples_frequence equals 0, which means the frequency can be set to any value between samples_frequence_min and samples_frequence_max.

Return esp_err_t

- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE USB device not active
- ESP_OK Success

Parameters

- `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

esp_err_t **uac_frame_size_reset**(*usb_stream_t stream*, uint8_t *ch_num*, uint16_t *bit_resolution*, uint32_t *samples_frequence*)

Reset audio channel number, bit resolution and samples frequence, please reset when the streaming in suspend state. The new configs will be effective after streaming resume.

Return 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

Parameters

- `stream`: stream type
- `ch_num`: audio channel numbers
- `bit_resolution`: audio bit resolution
- `samples_frequence`: audio samples frequence

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.

Return esp_err_t ESP_ERR_INVALID_ARG parameter error ESP_ERR_INVALID_STATE uvc stream not config or not active ESP_OK succeed

Parameters

- `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

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.

Return esp_err_t

Parameters

- `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

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 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 of MJPEG

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 behavers

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

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_frequence**
microphone frequence(Hz), UAC_FREQUENCY_ANY for any frequency

uint16_t **spk_bit_resolution**
speaker resolution(bits), UAC_BITS_ANY for any

uint32_t **spk_samples_frequence**
speaker frequence(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 behavers

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:

UVC_XFER_ISOC = 0
Isochronous Transfer Mode

UVC_XFER_BULK
Bulk Transfer Mode

UVC_XFER_UNKNOWN
Unknown Mode

enum usb_stream_t
Stream id, used for control.

Values:

STREAM_UVC = 0
usb video stream

STREAM_UAC_SPK
usb audio speaker stream

STREAM_UAC_MIC
usb audio microphone stream

STREAM_MAX
max stream id

enum usb_stream_state_t
USB device connection status.

Values:

STREAM_CONNECTED = 0

STREAM_DISCONNECTED

enum stream_ctrl_t
Stream control type, which also depends on if device support.

Values:

CTRL_NONE = 0
None

CTRL_SUSPEND
streaming suspend control. ctrl_data NULL

CTRL_RESUME
streaming resume control. ctrl_data NULL

CTRL_UAC_MUTE
mute control. ctrl_data (false/true)

CTRL_UAC_VOLUME
volume control. ctrl_data (0~100)

CTRL_MAX
max type value

CHAPTER

FIVE

AUDIO

[??]

5.1 PWM Audio

[??]

Supported Socs	ESP32	ESP32-S2	ESP32-S3	ESP32-C3

The PWM audio function uses the internal LEDC peripheral in ESP32 to generate PWM audio, which does not need an external audio Codec chip. This is mainly used for cost-sensitive applications with low audio quality requirements.

5.1.1 Features

- Allows any GIPO with output capability as an audio output pin
- Supports 8-bit ~ 10-bit PWM resolution
- Supports stereo
- Supports 8 ? 48 KHz sampling rate

5.1.2 Structure

1. First, the data is recoded to meet the PWM input requirements, including the shift and offset of the data;
2. Send data to the ISR (Interrupt Service Routines) function of the Timer Group via Ring Buffer;
3. The Timer Group reads data from the Ring Buffer according to the pre-defined sampling rate, and write the data into LEDC.

Note: Since the output is a PWM signal, it needs to be low-pass filtered to get the audio waveform.

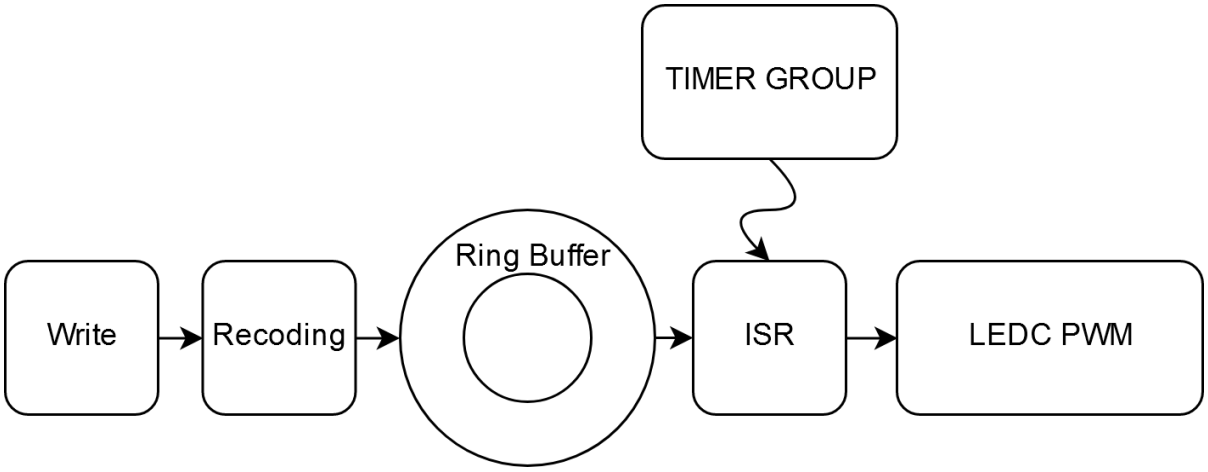


Fig. 1: Structure

5.1.3 PWM Frequency

The frequency of the output PWM cannot be configured directly, but needs to be calculated by configuring the number of PWM resolution bits, as shown below:

$$f_{pwm} = \frac{f_{APB\_CLK}}{2^{res\_bits}} - \left(\frac{f_{APB\_CLK}}{2^{res\_bits}} MOD 1000 \right)$$

The f_{APB_CLK} here is 80 MHz, and res_bits is the number of PWM resolution bits. When the resolution is `LEDC_TIMER_10_BIT`, the PWM frequency is 78 KHz. As we all know, a higher PWM frequency and resolution can better reproduce the audio signal. However, this formula shows that increasing PWM frequency means lower resolution and increasing resolution means lower PWM frequency. Thus, please adjust these parameters according to the actual application scenario to achieve a good balance.

5.1.4 Application Example

```
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␣
↪numner */
pwm_audio_start();                 /**< Start to run */

while(1) {

    //< Perpare audio data, such as decode mp3/wav file
```

(continues on next page)

(continued from previous page)

```
    /**< Write data to paly */
    pwm_audio_write(audio_data, length, &written, 1000 / portTICK_PERIOD_MS);
}
```

5.1.5 API Reference

Header File

- 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.

Return

- 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

Parameters

- `cfg`: configurations - see *pwm_audio_config_t* struct

esp_err_t **pwm_audio_deinit**(void)
Deinitialize LEDC timer_group and output gpio.

Return

- ESP_OK Success
- ESP_FAIL pwm_audio not initialized

esp_err_t **pwm_audio_start**(void)
Start to run pwm audio.

Return

- 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

Return

- 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.

Return

- ESP_OK Success
- ESP_FAIL Write encounter error
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `inbuf`: Pointer source data to write
- `len`: length of data in bytes
- `[out] bytes_written`: 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.

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.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `rate`: sample rate (ex: 8000, 44100…)
- `bits`: bit width
- `ch`: channel number

esp_err_t **pwm_audio_set_sample_rate**(int *rate*)
Set sample rate.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `rate`: sample rate (ex: 8000, 44100…)

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

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `volume`: Volume to set (-16 ~ 16). Set to 0 for original output; Set to less then 0 for attenuation, and -16 is mute; Set to more than 0 for enlarge, and 16 is double output

esp_err_t **pwm_audio_get_volume**(int8_t *volume)
Get current volume.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `volume`: Pointer to volume

esp_err_t **pwm_audio_get_param**(int *rate, int *bits, int *ch)
Get parameter for pwm audio.

Return

- Always return ESP_OK

Parameters

- `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

esp_err_t **pwm_audio_get_status**(*pwm_audio_status_t* *status)
get pwm audio status

Return

- ESP_OK Success

Parameters

- `status`: current pwm_audio status

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:

PWM_AUDIO_STATUS_UN_INIT = 0
pwm audio uninitialized

PWM_AUDIO_STATUS_IDLE = 1
pwm audio idle

PWM_AUDIO_STATUS_BUSY = 2
pwm audio busy

enum pwm_audio_channel_t
pwm audio channel define

Values:

PWM_AUDIO_CH_MONO = 0
1 channel (mono)

PWM_AUDIO_CH_STEREO = 1
2 channel (stereo)

PWM_AUDIO_CH_MAX

5.2 DAC Audio

[??]

ESP32 has two independent DAC channels and can play audio using I2S directly via DMA. The APIs in this document have been simplified on the basis of ESP-IDF, and the related data has been recoded to support more types of sampling bit width.

5.2.1 API Reference

Header File

- audio/dac_audio/include/dac_audio.h

Functions

esp_err_t **dac_audio_init**(*dac_audio_config_t* *cfg)
initialize i2s build-in dac to play audio with

Attention only support ESP32, because i2s of ESP32S2 not have a build-in dac

Return

- ESP_OK Success
- ESP_FAIL Encounter error
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM Out of memory

Parameters

- `cfg`: configurations - see *dac_audio_config_t* struct

esp_err_t **dac_audio_deinit**(void)
deinitialize dac

Return

- ESP_OK Success
- ESP_FAIL Encounter error

esp_err_t **dac_audio_start**(void)
Start dac to play.

Return

- ESP_OK Success
- ESP_FAIL Encounter error

esp_err_t **dac_audio_stop**(void)
Stop play.

Return

- ESP_OK Success
- ESP_FAIL Encounter error

esp_err_t **dac_audio_set_param**(int *rate*, int *bits*, int *ch*)
Configuration dac parameter.

Return

- ESP_OK Success
- ESP_FAIL Encounter error
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `rate`: sample rate (ex: 8000, 44100…)
- `bits`: bit width
- `ch`: channel number

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

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `volume`: Volume to set (-16 ~ 16), see Macro VOLUME_0DB Set to 0 for original output; Set to less then 0 for attenuation, and -16 is mute; Set to more than 0 for enlarge, and 16 is double output

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.

Return

- ESP_OK Success
- ESP_FAIL Write encounter error
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `inbuf`: Pointer source data to write
- `len`: length of data in bytes
- `[out] bytes_written`: 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.

Structures

struct dac_audio_config_t
Configuration parameters for dac_audio_init function.

Public Members

i2s_port_t **i2s_num**
I2S_NUM_0, I2S_NUM_1

int **sample_rate**
I2S sample rate

i2s_bits_per_sample_t **bits_per_sample**
I2S bits per sample

i2s_dac_mode_t **dac_mode**
DAC mode configurations - see i2s_dac_mode_t

int **dma_buf_count**
DMA buffer count, number of buffer

int **dma_buf_len**
DMA buffer length, length of each buffer

uint32_t **max_data_size**
one time max write data size

CHAPTER

SIX

GUI

[??]

6.1 LVGL Graphics Library

[??]

LVGL is an open-source free graphics library in C language providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

6.1.1 Features

LVGL has the following features:

- More than 30 powerful, fully customizable widgets, e.g., button, slider, text area, keyboard and so on
- Supports various screens with any resolution
- Simple interface and low memory usage
- Supports multiple input device for the same screen
- Provides various drawing features, e.g., anti-aliasing, polygon, shadow, etc.
- Supports UTF-8 coding, multi language and multi font text
- Supports various image formats, can read images from flash or SD card
- provides online image converter
- Supports Micropython

6.1.2 Requirements

The minimum requirements for running LVGL are listed as follows:

- 16, 32 or 64 bit micro-controller or processor
- Clock frequency: > 16 MHz
- Flash/ROM: > 64 kB (180 kB is recommended)
- RAM: 8 kB (24 kB is recommended)
- 1 frame buffer

- Graphics buffer: > “horizontal resolution” pixels
- C99 or newer compiler

6.1.3 Online Tools

LVGL provides online Font Converter and Image Converter.

6.1.4 Demo Examples

Note: The following examples are no longer maintained. For LCD and LVGL examples, please refer to: i80_controller rgb_panel And spi_lcd_touch

Official Demo

LVGL provides a demo project of using LVGL on ESP32 in LVGL project for ESP32.

On top of that, ESP-IoT-Solution also provides some application examples of using LVGL:

Thermostat

A thermostat control interface designed using LVGL:

Please find details of this example in hmi/lvgl_thermostat.

Coffee

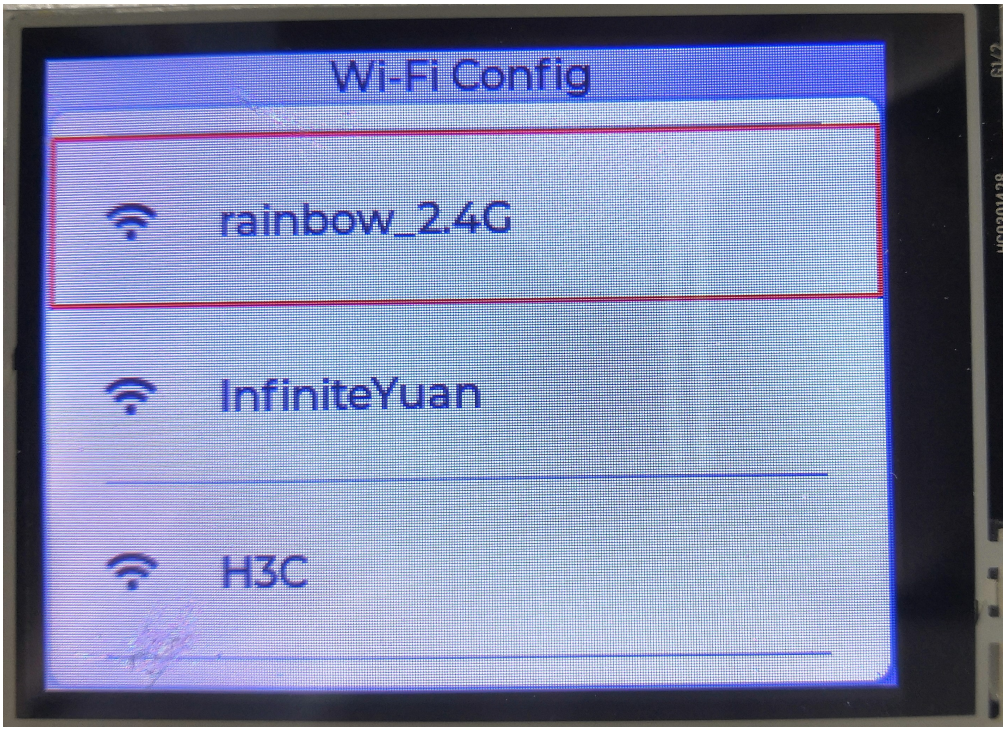
An interactive interface of a coffee machine designed using LVGL:

Please find details of this example in hmi/lvgl_coffee.

Wificonfig

When connecting Wi-Fi with ESP32, a Wi-Fi connection interface designed using LVGL can show information of the neighboring Wi-Fi, and you can type in Wi-Fi password on this interface.

Please find details of this example in hmi/lvgl_wificonfig.


Wi-Fi Config
rainbow_2.4G
InfiniteYuan
H3C


InfiniteYuan Password:
Canael
Connect
1# q w e r t y u i o p
ABC a s d f g h j k l
_ - z x c v b n m . , :

CHAPTER

SEVEN

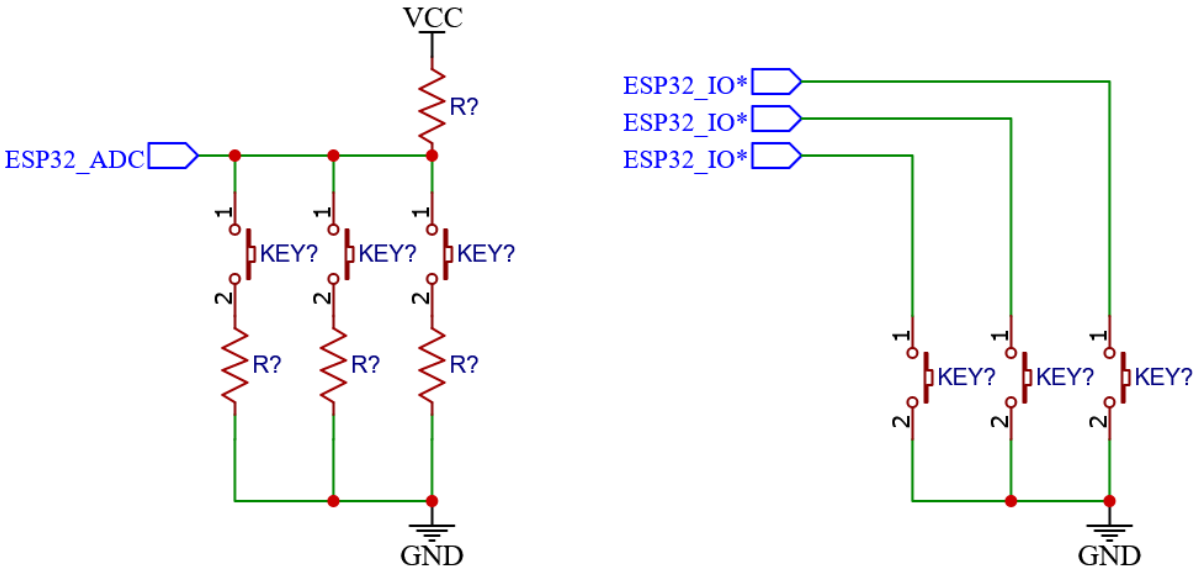
INPUT DEVICE

[??]

7.1 Button

[??]

The button component supports GPIO and ADC mode, and it allows the creation of two different kinds of the button at the same time. The following figure shows the hardware design of the button:



ADC Button GPIO Button

- GPIO button: The advantage of the GPIO button is that each button occupies an independent IO and therefore does not affect each other, and has high stability however when the number of buttons increases, it may take too many IO resources.
- ADC button: The advantage of using the ADC button is that one ADC channel can share multiple buttons and occupy fewer IO resources. The disadvantages include that you cannot press multiple buttons at the same time, and instability increases due to increase in the closing resistance of the button due to oxidation and other factors.

Note:

- The GPIO button needs to pay attention to the problem of pull-up and pull-down resistor inside the chip, which will be enabled by default. But there is no such resistor inside the IO that only supports input, **external connection requires**.
- The voltage of the ADC button should not exceed the ADC range.

7.1.1 Button event

Triggering conditions for each button event are enlisted in the table below:

Event	Trigger Conditions
BUTTON_PRESS_DOWN	Button press down
BUTTON_PRESS_UP	Button release
BUTTON_PRESS_REPEAT	Button press down and release (>= 2-times)
BUTTON_SINGLE_CLICK	Button press down and release (single time)
BUTTON_DOUBLE_CLICK	Button press down and release (double times)
BUTTON_LONG_PRESS_START	Button press reaches the long-press threshold
BUTTON_LONG_PRESS_HOLD	Button press for long time
BUTTON_PRESS_REPEAT_DONE	Multiple press down and release

Each button supports **call-back** and **pooling** mode.

- Call-back: Each event of a button can register a call-back function for it, and the call-back function will be called when an event is generated. This method has high efficiency and real-time performance, and no events will be lost.
- Polling: Periodically call `iot_button_get_event()` in the program to query the current event of the button. This method is easy to use and is suitable for occasions with simple tasks

Note: you can also combine the above two methods.

Attention: No blocking operations such as **TaskDelay** are allowed in the call-back function

7.1.2 Configuration

- BUTTON_PERIOD_TIME_MS : scan cycle
- BUTTON_DEBOUNCE_TICKS : debounce time
- BUTTON_SHORT_PRESS_TIME_MS : short press down effective time
- BUTTON_LONG_PRESS_TIME_MS : long press down effective time
- ADC_BUTTON_MAX_CHANNEL : maximum number of channel for ADC
- ADC_BUTTON_MAX_BUTTON_PER_CHANNEL : maximum number of ADC buttons per channel
- ADC_BUTTON_SAMPLE_TIMES : ADC sample time
- BUTTON_SERIAL_TIME_MS : call-back interval triggered by long press time

7.1.3 Demonstration

Create a button

```
// create gpio button
button_config_t gpio_btn_cfg = {
    .type = BUTTON_TYPE_GPIO,
    .long_press_ticks = CONFIG_BUTTON_LONG_PRESS_TIME_MS,
    .short_press_ticks = CONFIG_BUTTON_SHORT_PRESS_TIME_MS,
    .gpio_button_config = {
        .gpio_num = 0,
        .active_level = 0,
    },
};
button_handle_t gpio_btn = iot_button_create(&gpio_btn_cfg);
if(NULL == gpio_btn) {
    ESP_LOGE(TAG, "Button create failed");
}

// create adc button
button_config_t adc_btn_cfg = {
    .type = BUTTON_TYPE_ADC,
    .long_press_ticks = CONFIG_BUTTON_LONG_PRESS_TIME_MS,
    .short_press_ticks = CONFIG_BUTTON_SHORT_PRESS_TIME_MS,
    .adc_button_config = {
        .adc_channel = 0,
        .button_index = 0,
        .min = 100,
        .max = 400,
    },
};
button_handle_t adc_btn = iot_button_create(&adc_btn_cfg);
if(NULL == adc_btn) {
    ESP_LOGE(TAG, "Button create failed");
}
```

Note: please pass adc_handle and adc_channel , when there are other places in the project that use ADC1.

Register call-back

```
static void button_single_click_cb(void *arg,void *usr_data)
{
    ESP_LOGI(TAG, "BUTTON_SINGLE_CLICK");
}

iot_button_register_cb(gpio_btn, BUTTON_SINGLE_CLICK, button_single_click_cb,NULL);
```

Find an event

```
button_event_t event;
event = iot_button_get_event(button_handle);
```

7.1.4 API Reference

Header File

- button/include/iot_button.h

Functions

button_handle_t **iot_button_create**(**const** *button_config_t* **config*)
Create a button.

Return A handle to the created button, or NULL in case of error.

Parameters

- `config`: pointer of button configuration, must corresponding the button type

esp_err_t **iot_button_delete**(*button_handle_t btn_handle*)
Delete a button.

Return

- ESP_OK Success
- ESP_FAIL Failure

Parameters

- `btn_handle`: A button handle to delete

esp_err_t **iot_button_register_cb**(*button_handle_t btn_handle*, *button_event_t event*, *button_cb_t cb*, void **usr_data*)
Register the button event callback function.

Return

- 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.

Parameters

- `btn_handle`: A button handle to register
- `event`: Button event
- `cb`: Callback function.
- `usr_data`: user data

esp_err_t **iot_button_unregister_cb**(*button_handle_t btn_handle*, *button_event_t event*)
Unregister the button event callback function.

Return

- 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.

Parameters

- `btn_handle`: A button handle to unregister
- `event`: Button event

size_t **iot_button_count_cb**(*button_handle_t btn_handle*)
how many Callbacks are still registered.

Return 0 if no callbacks registered, or 1 .. (BUTTON_EVENT_MAX-1) for the number of Registered Buttons.

Parameters

- `btn_handle`: A button handle to unregister

button_event_t **iot_button_get_event**(*button_handle_t btn_handle*)
Get button event.

Return Current button event. See button_event_t

Parameters

- `btn_handle`: Button handle

uint8_t **iot_button_get_repeat**(*button_handle_t btn_handle*)
Get button repeat times.

Return button pressed times. For example, double-click return 2, triple-click return 3, etc.

Parameters

- `btn_handle`: Button handle

uint16_t **iot_button_get_ticks_time**(*button_handle_t btn_handle*)
Get button ticks time.

Return Actual time from press down to up (ms).

Parameters

- `btn_handle`: Button handle

uint16_t **iot_button_get_long_press_hold_cnt**(*button_handle_t btn_handle*)
Get button long press hold count.

Return Count of trigger cb(BUTTON_LONG_PRESS_HOLD)

Parameters

- `btn_handle`: Button handle

Structures

struct button_custom_config_t
custom button configuration

Public Members

uint8_t **active_level**
active level when press down

esp_err_t (***button_custom_init**)(void *param)
user defined button init

uint8_t (***button_custom_get_key_value**)(void *param)
user defined button get key value

esp_err_t (***button_custom_deinit**)(void *param)
user defined button deinit

void ***priv**
private data used for custom button, MUST be allocated dynamically and will be auto freed in iot_button_delete

struct button_config_t
Button configuration.

Public Members

button_type_t **type**
button type, The corresponding button configuration must be filled

uint16_t **long_press_time**
Trigger time(ms) for long press, if 0 default to BUTTON_LONG_PRESS_TIME_MS

uint16_t **short_press_time**
Trigger time(ms) for short press, if 0 default to BUTTON_SHORT_PRESS_TIME_MS

button_gpio_config_t **gpio_button_config**
gpio button configuration

button_adc_config_t **adc_button_config**
adc button configuration

button_custom_config_t **custom_button_config**
custom button configuration

union *button_config_t*::**[anonymous] [anonymous]**
button configuration

Type Definitions

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

typedef void ***button_handle_t**

Enumerations

enum button_event_t

Button events.

Values:

BUTTON_PRESS_DOWN = 0

BUTTON_PRESS_UP

BUTTON_PRESS_REPEAT

BUTTON_PRESS_REPEAT_DONE

BUTTON_SINGLE_CLICK

BUTTON_DOUBLE_CLICK

BUTTON_LONG_PRESS_START

BUTTON_LONG_PRESS_HOLD

BUTTON_EVENT_MAX

BUTTON_NONE_PRESS

enum button_type_t

Supported button type.

Values:

BUTTON_TYPE_GPIO

BUTTON_TYPE_ADC

BUTTON_TYPE_CUSTOM

7.2 Knob

[English]

Knob is the component that provides the software PCNT, it can be used on chips(esp32c2, esp32c3) that do not have PCNT hardware capabilities. By using knob you can quickly use a physical encoder, such as the EC11 encoder.

7.2.1 Applicable Scenarios

This is suitable for low-speed rotary knob counting scenarios where the pulse rate is less than 30 pulses per second, such as the EC11 encoder. It is suitable for scenarios where 100% accuracy of pulse counting is not required.

Note: For precise or fast pulse counting, please use the hardware PCNT function. The hardware PCNT is supported by ESP32, ESP32-C6, ESP32-H2, ESP32-S2, ESP32-S3 chips.

7.2.2 Knob Event

Each knob has the 5 events in the following table.

Events	Trigger Conditions
KNOB_LEFT	Left
KNOB_RIGHT	Right
KNOB_H_LIM	Count reaches maximum limit
KNOB_L_LIM	Count reaches the minimum limit
KNOB_ZERO	Count back to 0

Each knob can have **Callback** usage.

- Callbacks: Each event of a knob can have a callback function registered for it, and the callback function will be called when the event is generated. This approach is efficient and real-time, and no events are lost.

Attention: No blocking operations such as TaskDelay in the callback function

7.2.3 Configuration items

- KNOB_PERIOD_TIME_MS : Scan cycle
- KNOB_DEBOUNCE_TICKS : Number of de-shaking
- KNOB_HIGH_LIMIT : The highest number that can be counted by the knob
- KNOB_LOW_LIMIT : The lowest number that can be counted by the knob

7.2.4 Application Examples

Create Knob

```
// 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");
}
```

Register callback function

```
static void _knob_left_cb(void *arg, void *data)
{
    ESP_LOGI(TAG, "KONB: KONB_LEFT,count_value:%"PRId32"",iot_knob_get_count_
↪value((button_handle_t)arg));
}
iot_knob_register_cb(s_knob, KNOB_LEFT, _knob_left_cb, NULL);
```

7.2.5 API Reference

Header File

- knob/iot_knob.h

Functions

knob_handle_t **iot_knob_create**(**const** *knob_config_t* **config*)
create a knob

Return A handle to the created knob

Parameters

- `config`: pointer of knob configuration

esp_err_t **iot_knob_delete**(*knob_handle_t knob_handle*)
Delete a knob.

Return

- ESP_OK Success
- ESP_FAIL Failure

Parameters

- `knob_handle`: A knob handle to delete

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.

Return

- ESP_OK Success
- ESP_FAIL Failure

Parameters

- `knob_handle`: A knob handle to register
- `event`: Knob event
- `cb`: Callback function
- `usr_data`: user data

esp_err_t **iot_knob_unregister_cb**(*knob_handle_t knob_handle*, *knob_event_t event*)
Unregister the knob event callback function.

Return

- ESP_OK Success
- ESP_FAIL Failure

Parameters

- `knob_handle`: A knob handle to register
- `event`: Knob event

knob_event_t **iot_knob_get_event**(*knob_handle_t knob_handle*)
Get knob event.

Return knob_event_t Knob event

Parameters

- `knob_handle`: A knob handle to register

int **iot_knob_get_count_value**(*knob_handle_t knob_handle*)
Get knob count value.

Return int count_value

Parameters

- `knob_handle`: A knob handle to register

esp_err_t **iot_knob_clear_count_value**(*knob_handle_t knob_handle*)
Clear knob cout value to zero.

Return

- ESP_OK Success
- ESP_FAIL Failure

Parameters

- `knob_handle`: A knob handle to register

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

Type Definitions

typedef void (***knob_cb_t**) (void *, void *)

typedef void ***knob_handle_t**

Enumerations

enum knob_event_t
Knob events.

Values:

KNOB_LEFT = 0
EVENT: Rotate to the left

KNOB_RIGHT
EVENT: Rotate to the right

KNOB_H_LIM
EVENT: Count reaches maximum limit

KNOB_L_LIM
EVENT: Count reaches the minimum limit

KNOB_ZERO
EVENT: Count back to 0

KNOB_EVENT_MAX
EVENT: Number of events

7.3 Touch Panel

[??]

Touch panels are now standard components in display applications. ESP-IoT-Solution provides drivers for common types of touch panels and currently supports the following controllers:

Resistive Touch Panel	Capacitive Touch Panel
XPT2046	FT5216
NS2016	FT5436
	FT6336
	FT5316

The capacitive touch panel controllers listed above can usually be driven by `FT5x06`.

Similar to the screen driver, some common functions are encapsulated in the *touch_panel_driver_t* structure, in order to port them to different GUI libraries easily. After initializing the touch panel, users can conduct operations by calling functions inside the structure, without paying attention to specific touch panel models.

7.3.1 Touch Panel Calibration

In actual applications, resistive touch panels must be calibrated before use, while capacitive touch panels are usually calibrated by controllers and do not require extra calibration steps. A calibration algorithm is integrated in the resistive touch panel driver. During the process, three points are used to calibrate, in which one point is used for verification. If the verified error exceeds a certain threshold value, it means the calibration has failed and a new round of calibration is started automatically until it succeeds.

The calibration process will be started by calling `calibration_run()`. After finished, the parameters are stored in NVS for next initialization to avoid repetitive work.

7.3.2 Press Touch Panel

Generally, there is an interrupt pin inside the touch panel controller (both resistive and capacitive) to signal touch events. However, this is not used in the touch panel driver, because IOs should be saved for other peripherals in screen applications as much as possible; on the other hand the information in this signal is not as accurate as data in registers.

For resistive touch panels, when the pressure in the Z direction exceeds the configured threshold, it is considered as pressed; for capacitive touch panels, the detection of over one touch point will be considered as pressed.

7.3.3 Touch Panel Rotation

A touch panel has eight directions, like the screen, defined in *touch_panel_dir_t*. The rotation of a touch panel is achieved by software, which usually sets the direction of a touch panel and a screen as the same. But this should not be fixed, for example, when using a capacitive touch panel, the inherent direction of the touch panel may not fit with the original display direction of the screen. Simply setting these two directions as the same may not show the desired contents. Therefore, please adjust the directions according to the actual situation.

On top of that, the configuration of its resolution is also important since the converted display after a touch panel being rotated relies on the resolution of its width and height. An incorrect configuration of the resolution may give you a distorted display.

Note: If you are using a resistive touch panel, the touch position can become inaccurate after it being rotated, since the resistance value in each direction may not be distributed uniformly. It is recommended to not rotate a resistive touch panel after it being calibrated.

7.3.4 Application Example

Initialize a Touch Panel

```
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);
```

Note:

- When using a capacitive touch panel, the call to the calibration function will return `ESP_OK` directly.
- By default, only FT5x06 touch panel driver is enabled, please go to `menuconfig -> Component config -> Touch Screen Driver -> Choose Touch Screen Driver` to do configurations if you need to enable other drivers.

To Know If a Touch Panel is Pressed and Its Corresponding Position

```
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);
}
```

7.3.5 API Reference

Header File

- 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.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG Arguments is NULL.
- ESP_ERR_NOT_FOUND: Touch panel controller was not found.

Parameters

- `controller`: Touch panel controller to initialize
- `out_driver`: Pointer to a touch driver

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 adddress

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.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `config`: Pointer to a structure with touch config arguments.

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.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `screen`: Screen driver for display prompts
- `recalibrate`: Is mandatory, set true to force calibrate

esp_err_t (*__set_direction__)(*touch_panel_dir_t* dir)
Set touch rotate rotation.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `dir`: rotate direction

esp_err_t (*__read_point_data__)(*touch_panel_points_t* *point)
Get current touch information, see struct *touch_panel_points_t*.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `point`: a pointer of *touch_panel_points_t* contained touch information.

Macros

TOUCH_MAX_POINT_NUMBER
max point number on touch panel

Enumerations

enum touch_panel_event_t
Touch events.

Values:

TOUCH_EVT_RELEASE = 0x0
Release event

TOUCH_EVT_PRESS = 0x1
Press event

enum touch_panel_dir_t
Define all screen direction.

Values:

TOUCH_DIR_LRTB
From left to right then from top to bottom, this consider as the original direction of the touch panel

TOUCH_DIR_LRBT
From left to right then from bottom to top

TOUCH_DIR_RLTB
From right to left then from top to bottom

TOUCH_DIR_RLBT
From right to left then from bottom to top

TOUCH_DIR_TBLR
From top to bottom then from left to right

TOUCH_DIR_BTLR
From bottom to top then from left to right

TOUCH_DIR_TBRL
From top to bottom then from right to left

TOUCH_DIR_BTRL
From bottom to top then from right to left

TOUCH_DIR_MAX

TOUCH_MIRROR_X = 0x40
Mirror X-axis

TOUCH_MIRROR_Y = 0x20
Mirror Y-axis

TOUCH_SWAP_XY = 0x80
Swap XY axis

enum touch_panel_interface_type_t
Values:

TOUCH_PANEL_IFACE_I2C
I2C interface

TOUCH_PANEL_IFACE_SPI
SPI interface

enum touch_panel_controller_t
All supported touch panel controllers.

Values:

TOUCH_PANEL_CONTROLLER_FT5X06

TOUCH_PANEL_CONTROLLER_XPT2046

TOUCH_PANEL_CONTROLLER_NS2016

CHAPTER

EIGHT

SENSORS

[??]

8.1 Sensor Hub

[??]

Sensor Hub is a sensor management component that can realize hardware abstraction, device management and data distribution for sensor devices. When developing applications based on Sensor Hub, users do not have to deal with complex sensor implementations, but only need to make simple selections for sensor operation, acquisition interval, range, etc. and then register callback functions to the event messages of your interests. By doing so, users can receive notifications when sensor states are switched or when data is collected.

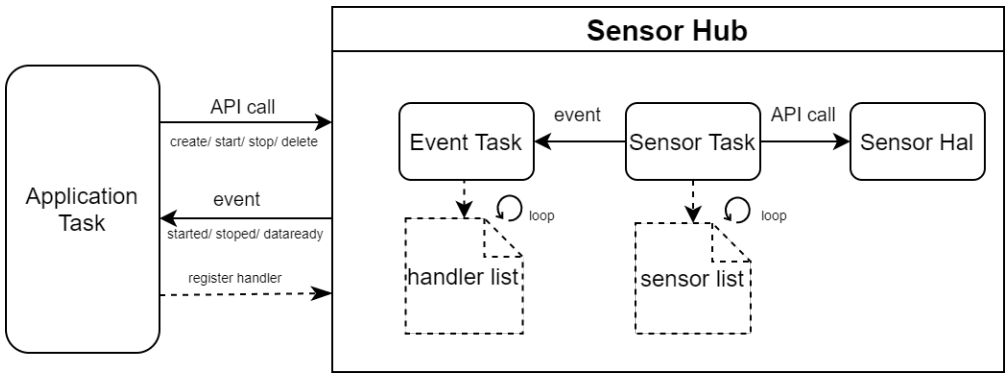


Fig. 1: Sensor Hub Programming Model

The Sensor Hub provides hardware abstraction for common sensor categories, based on which users can switch sensor models without modifying the upper layer of the application. And it allows adding new sensors to the Sensor Hub by implementing their corresponding sensor interface at hardware abstraction layer. This component can be used as a basic component for sensor applications in various intelligent scenarios such as environment monitoring, motion detection, health management and etc. as it simplifies operation and improves operating efficiency by centralized management of sensors.

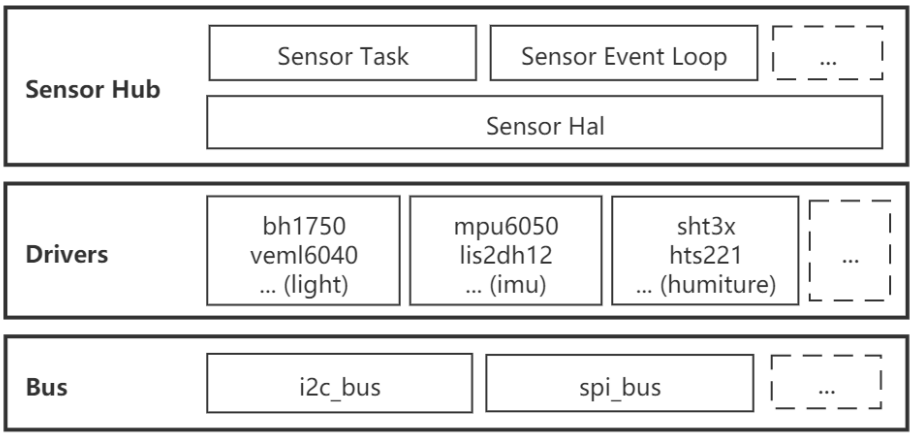


Fig. 2: Sensor Hub Driver

8.1.1 Instructions

1. Create a sensor instance: use *iot_sensor_create()* to create a sensor instance. The related parameters include the sensor ID defined in `sensor_id_t`, configuration options for the sensor and its handler pointer. The sensor ID is used to find and load the corresponding driver, and each ID can only be used for one sensor instance. In configuration options, *bus* is used to specify the bus location on which the sensor is mounted; *mode* is used to specify the operating mode of the sensor; *min_delay* is used to specify the acquisition interval of the sensor, while other items inside are all non-required options. After the instance is created, the sensor handler is obtained;
2. Register callback functions for sensor events: when a sensor event occurs, the callback functions will be called in sequence. There are two ways to register a callback function, and the instance handler of the event callback function will be returned after the registration succeed:
   - Use *iot_sensor_handler_register()* to register a callback function with the sensor handler
   - Use *iot_sensor_handler_register_with_type()* to register a callback function with the sensor type
3. Start a sensor: use *iot_sensor_start()* to start a specific sensor. After started, it will trigger a `SENSOR_STARTED` event, then it will collect the sensor data continuously with a set of period and trigger `SENSOR_XXXX_DATA_READY` event. The event callback function can obtain the specific data of each event via the `event_data` parameter;
4. Stop a sensor: use *iot_sensor_stop()* to stop a specified sensor temporarily. After stopped, the sensor will send out a `SENSOR_STOPED` event and then stop the data collecting work. If the driver of this sensor supports power management, the sensor will be set to sleep mode in this stage;
5. Unregister callback functions for sensor events: the user program can unregister an event at any time using the instance handler of this event callback function, and this callback function will not be called again when this event occurs afterwards. There are also two ways to do so:
   - Use *iot_sensor_handler_unregister()* to unregister the callback function with the sensor handler
   - Use *iot_sensor_handler_unregister_with_type()* to unregister the callback function with the sensor type

6. Delete sensors: use *iot_sensor_delete()* to delete the corresponding sensor to release the allocated memory and other resources.

8.1.2 Examples

1. Sensor control LED example: sensors/sensor_control_led.
2. Sensor hub monitor example: sensors/sensor_hub_monitor.

8.1.3 API Reference

Header File

- sensors/sensor_hub/include/sensor_type.h

Structures

struct sensor_data_t
sensor data type

Public Members

int64_t **timestamp**
timestamp

uint8_t **sensor_id**
sensor id

int32_t **event_id**
reserved for future use

uint32_t **min_delay**
minimum delay between two events, unit: ms

axis3_t **acce**
Accelerometer. unit: G

axis3_t **gyro**
Gyroscope. unit: dps

axis3_t **mag**
Magnetometer. unit: Gauss

float **temperature**
Temperature. unit: dCelsius

float **humidity**
Relative humidity. unit: percentage

float **baro**
Pressure. unit: pascal (Pa)

float **light**
Light. unit: lux

rgbw_t **rgbw**
Color. unit: lux

uv_t **uv**
ultraviole unit: lux

float **proximity**
Distance. unit: centimeters

float **hr**
Heat rate. unit: HZ

float **tvoc**
TVOC. unit: permillage

float **noise**
Noise Loudness. unit: HZ

float **step**
Step sensor. unit: 1

float **force**
Force sensor. unit: mN

float **current**
Current sensor unit: mA

float **voltage**
Voltage sensor unit: mV

float **data**[4]
for general use

struct sensor_data_group_t
sensor data group type

Public Members

uint8_t **number**
effective data number

sensor_data_t **sensor_data**[**SENSOR_DATA_GROUP_MAX_NUM**]
data buffer

Macros

SENSOR_EVENT_ANY_ID
register handler for any event id

Type Definitions

typedef void ***sensor_driver_handle_t**
hal level sensor driver handle

typedef void ***bus_handle_t**
i2c/spi bus handle

Enumerations

enum sensor_type_t
sensor type

Values:

NULL_ID
NULL

HUMITURE_ID
humidity or temperature sensor

IMU_ID
gyro or acc sensor

LIGHT_SENSOR_ID
light illumination or uv or color sensor

SENSOR_TYPE_MAX
max sensor type

enum sensor_command_t
sensor operate command

Values:

COMMAND_SET_MODE
set measure mdoe

COMMAND_SET_RANGE
set measure range

COMMAND_SET_ODR
set output rate

COMMAND_SET_POWER
set power mode

COMMAND_SELF_TEST
sensor self test

COMMAND_MAX
max sensor command

enum sensor_power_mode_t
sensor power mode

Values:

POWER_MODE_WAKEUP
wakeup from sleep

POWER_MODE_SLEEP
set to sleep

POWER_MAX
max sensor power mode

enum sensor_mode_t
sensor acquire mode

Values:

MODE_DEFAULT
default work mode

MODE_POLLING
polling acquire with a interval time

MODE_INTERRUPT
interrupt mode, acquire data when interrupt comes

MODE_MAX
max sensor mode

enum sensor_range_t
sensor acquire range

Values:

RANGE_DEFAULT
default range

RANGE_MIN
minimum range for high-speed or high-precision

RANGE_MEDIUM
medium range for general use

RANGE_MAX
maximum range for full scale

enum sensor_event_id_t
sensor general events

Values:

SENSOR_STARTED
sensor started, data acquire will be started

SENSOR_STOPED
sensor stoped, data acquire will be stoped

SENSOR_EVENT_COMMON_END = 9
max common events id

enum sensor_data_event_id_t
sensor data ready events

Values:

SENSOR_ACCE_DATA_READY = 10
Accelerometer data ready

SENSOR_GYRO_DATA_READY
Gyroscope data ready

SENSOR_MAG_DATA_READY
Magnetometer data ready

SENSOR_TEMP_DATA_READY
Temperature data ready

SENSOR_HUMI_DATA_READY
Relative humidity data ready

SENSOR_BARO_DATA_READY
Pressure data ready

SENSOR_LIGHT_DATA_READY
Light data ready

SENSOR_RGBW_DATA_READY
Color data ready

SENSOR_UV_DATA_READY
ultraviolet data ready

SENSOR_PROXI_DATA_READY
Distance data ready

SENSOR_HR_DATA_READY
Heat rate data ready

SENSOR_TVOC_DATA_READY
TVOC data ready

SENSOR_NOISE_DATA_READY
Noise Loudness data ready

SENSOR_STEP_DATA_READY
Step data ready

SENSOR_FORCE_DATA_READY
Force data ready

SENSOR_CURRENT_DATA_READY
Current data ready

SENSOR_VOLTAGE_DATA_READY
Voltage data ready

SENSOR_EVENT_ID_END
max common events id

Header File

- sensors/sensor_hub/include/iot_sensor_hub.h

Functions

esp_err_t **iot_sensor_create**(*sensor_id_t* *sensor_id*, **const** *sensor_config_t* **config*, *sensor_handle_t* **p_sensor_handle*)
Create a sensor instance with specified sensor_id and desired configurations.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_id`: sensor's id detailed in sensor_id_t.
- `config`: sensor's configurations detailed in *sensor_config_t*
- `p_sensor_handle`: return sensor handle if succeed, NULL if failed.

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.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_handle`: sensor handle for operation

esp_err_t **iot_sensor_stop** (*sensor_handle_t sensor_handle*)
stop sensor acquisition, and stop post data events. if stop succeed, SENSOR_STOPED event will be posted.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_handle`: sensor handle for operation

esp_err_t **iot_sensor_delete** (*sensor_handle_t *p_sensor_handle*)
delete and release the sensor resource.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `p_sensor_handle`: point to sensor handle, will set to NULL if delete suceed.

uint8_t **iot_sensor_scan** (*bus_handle_t bus*, *sensor_info_t *buf*[], uint8_t *num*)
Scan for valid sensors attached on bus.

Return uint8_t total number of valid sensors found on the bus

Parameters

- `bus`: bus handle
- `buf`: Pointer to a buffer to save sensors' information, if NULL no information will be saved.
- `num`: Maximum number of sensor information to save, invalid if buf set to NULL, latter sensors will be discarded if num less-than the total number found on the bus.

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.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `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.

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.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_handle`: sensor handle for operation
- `context`: the instance object of the registration to be unregistered

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.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_type`: sensor type decleared in sensor_type_t.
- `event_id`: sensor event decleared 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.

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.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor_type`: sensor type decleared in sensor_type_t.
- `event_id`: sensor event decleared in sensor_event_id_t and sensor_data_event_id_t
- `context`: the instance object of the registration to be unregistered

Structures

struct sensor_info_t

sensor information type

Public Members

const char ***name**

sensor name

const char ***desc**

sensor descriptive message

sensor_id_t **sensor_id**

sensor id

const uint8_t ***addrs**

sensor address list

struct sensor_config_t

sensor initialization parameter

Public Members

bus_handle_t **bus**

i2c/spi bus handle

sensor_mode_t **mode**

set acquire mode detiled in sensor_mode_t

sensor_range_t **range**

set measuring range

uint32_t **min_delay**

set minimum acquisition interval

int **intr_pin**

set interrupt pin

int **intr_type**
set interrupt type

Type Definitions

typedef void ***sensor_handle_t**
sensor handle

typedef void ***sensor_event_handler_instance_t**
sensor event handler handle

typedef const char ***sensor_event_base_t**
unique pointer to a subsystem that exposes events

typedef void (***sensor_event_handler_t**) (void *event_handler_arg, *sensor_event_base_t* event_base, int32_t event_id, void *event_data)
function called when an event is posted to the queue

Enumerations

enum sensor_id_t
sensor id, used for iot_sensor_create

Values:

8.2 Humidity and Temperature Sensor

[??]

The humidity and temperature sensor can be used as a temperature sensor, a humidity sensor or a sensor with both functions. It is mainly used for environmental temperature and humidity detections in smart home, smart farm and smart factory applications.

8.2.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	HAL
HDC2010	Temperature, Humidity	I2C	TI	Datasheet	
HTS221	Temperature, Humidity	I2C	ST	Datasheet	√
SHT3X	Temperature, Humidity	I2C	Sensirion	Datasheet	√
MVH3004D	Temperature, Humidity	I2C	–		

8.2.2 API Reference

The following APIs have implemented hardware abstraction on the humidity and temperature sensor. Users can call the code from this layer directly to write a sensor application, or use the sensor interface in *sensor_hub* for easier development.

Header File

- sensors/sensor_hub/include/hal/humiture_hal.h

Functions

sensor_humiture_handle_t **humiture_create** (*bus_handle_t bus*, int *id*)
Create a humiture/temperature/humidity sensor instance. Same series' sensor or sensor with same address can only be created once.

Return sensor_humiture_handle_t return humiture sensor handle if succeed, return NULL if create failed.

Parameters

- `bus`: i2c bus handle the sensor attached to
- `id`: id declared in humiture_id_t

esp_err_t **humiture_delete** (*sensor_humiture_handle_t* **sensor*)
Delete and release the sensor resource.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: point to humiture sensor handle, will set to NULL if delete succeed.

esp_err_t **humiture_test** (*sensor_humiture_handle_t sensor*)
Test if sensor is active.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: humiture sensor handle to operate

esp_err_t **humiture_acquire_humidity** (*sensor_humiture_handle_t sensor*, float **humidity*)
Acquire humiture sensor relative humidity result one time.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: humiture sensor handle to operate.
- `humidity`: result data (unit:percentage)

esp_err_t **humiture_acquire_temperature**(*sensor_humiture_handle_t sensor*, float **sensor_data*)
Acquire humiture sensor temperature result one time.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: humiture sensor handle to operate.
- `sensor_data`: result data (unit:dCelsius)

esp_err_t **humiture_sleep**(*sensor_humiture_handle_t sensor*)
Set sensor to sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: humiture sensor handle to operate

esp_err_t **humiture_wakeup**(*sensor_humiture_handle_t sensor*)
Wakeup sensor from sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: humiture sensor handle to operate

esp_err_t **humiture_acquire**(*sensor_humiture_handle_t sensor*, *sensor_data_group_t *data_group*)
acquire a group of sensor data

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: humiture sensor handle to operate

- `data_group`: acquired data

esp_err_t **`humiture_control`** (*sensor_humiture_handle_t* *sensor*, *sensor_command_t* *cmd*, void **args*)
control sensor mode with control commands and args

Parameters

- `sensor`: humiture sensor handle to operate
- `cmd`: control commands detailed in sensor_command_t
- `args`: control commands args
  - ESP_OK Success
  - ESP_FAIL Fail
  - ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Type Definitions

`typedef` void ***`sensor_humiture_handle_t`**
humiture sensor handle

Enumerations

`enum humiture_id_t`
humiture sensor id, used for humiture_create

Values:

`SHT3X_ID` = 0x01
sht3x humiture sensor id

`HTS221_ID`
hts221 humiture sensor id

`HUMITURE_MAX_ID`
max humiture sensor id

8.3 Inertial Measurement Unit (IMU)

[??]

The Inertial Measurement Unit (IMU) can be used as a gyroscope sensor, an acceleration sensor, a sensor with multiple functions or etc. It is mainly used to measure the acceleration and angular velocity of an object, and then calculate the motion attitude of the object.

8.3.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	HAL
LIS2DH12	3-axis acceler	I2C	ST	Datasheet	√
MPU6050	3-axis acceler + 3-axis gyro	I2C	InvenSense	Datasheet	√

8.3.2 API Reference

The following APIs have implemented hardware abstraction on the IMU. Users can call the code from this layer directly to write a sensor application, or use the sensor interface in *sensor_hub* for easier development.

Header File

- sensors/sensor_hub/include/hal/imu_hal.h

Functions

sensor_imu_handle_t **imu_create** (*bus_handle_t bus*, int *imu_id*)

Create a Inertial Measurement Unit sensor instance. Same series' sensor or sensor with same address can only be created once.

Return sensor_imu_handle_t return imu sensor handle if succeed, NULL is failed.

Parameters

- `bus`: i2c bus handle the sensor attached to
- `imu_id`: id declared in imu_id_t

esp_err_t **imu_delete** (*sensor_imu_handle_t *sensor*)

Delete and release the sensor resource.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: point to imu sensor handle, will set to NULL if delete succeed.

esp_err_t **imu_test** (*sensor_imu_handle_t sensor*)

Test if sensor is active.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: imu sensor handle to operate

esp_err_t **imu_acquire_acce**(*sensor_imu_handle_t sensor*, axis3_t **acce*)
Acquire imu sensor accelerometer result one time.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: imu sensor handle to operate
- `acce`: result data (unit:g)

esp_err_t **imu_acquire_gyro**(*sensor_imu_handle_t sensor*, axis3_t **gyro*)
Acquire imu sensor gyroscope result one time.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: imu sensor handle to operate
- `gyro`: result data (unit:dps)

esp_err_t **imu_sleep**(*sensor_imu_handle_t sensor*)
Set sensor to sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: imu sensor handle to operate

esp_err_t **imu_wakeup**(*sensor_imu_handle_t sensor*)
Wakeup sensor from sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: imu sensor handle to operate

esp_err_t **imu_acquire**(*sensor_imu_handle_t sensor*, *sensor_data_group_t *data_group*)
acquire a group of sensor data

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: imu sensor handle to operate
- `data_group`: acquired data

esp_err_t **imu_control**(*sensor_imu_handle_t sensor*, *sensor_command_t cmd*, void *args)
control sensor mode with control commands and args

Parameters

- `sensor`: imu sensor handle to operate
- `cmd`: control commands detailed in sensor_command_t
- `args`: control commands args
  - ESP_OK Success
  - ESP_FAIL Fail
  - ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Type Definitions

typedef void ***sensor_imu_handle_t**
imu sensor handle

Enumerations

enum imu_id_t
imu sensor id, used for imu_create

Values:

MPU6050_ID = 0x01
MPU6050 imu sensor id

LIS2DH12_ID
LIS2DH12 imu sensor id

IMU_MAX_ID
max imu sensor id

8.4 Ambient Light Sensor

[??]

The ambient light sensor can be used as a light intensity sensor, a color sensor, a UV sensor or a sensor with multiple functions.

8.4.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	HAL
BH1750	Light	I2C	rohm	Datasheet	√
VEML6040	Light RGBW	I2C	Vishay	Datasheet	√
VEML6075	Light UVA UVB	I2C	Vishay	Datasheet	√

8.4.2 API Reference

The following APIs have implemented hardware abstraction on the ambient light sensor. Users can call the code from this layer directly to write a sensor application, or use the sensor interface in *sensor_hub* for easier development.

Header File

- sensors/sensor_hub/include/hal/light_sensor_hal.h

Functions

sensor_light_handle_t **light_sensor_create** (*bus_handle_t bus*, int *id*)

Create a light sensor instance. same series' sensor or sensor with same address can only be created once.

Return sensor_light_handle_t return light sensor handle if succeed, return NULL if failed.

Parameters

- `bus`: i2c bus handle the sensor attached to
- `id`: id declared in light_sensor_id_t

esp_err_t **light_sensor_delete** (*sensor_light_handle_t* **sensor*)

Delete and release the sensor resource.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: point to light sensor handle, will set to NULL if delete succeed.

esp_err_t **light_sensor_test** (*sensor_light_handle_t sensor*)

Test if sensor is active.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: light sensor handle to operate.

esp_err_t **light_sensor_acquire_light**(*sensor_light_handle_t sensor*, float *lux)
Acquire light sensor illuminance result one time.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: light sensor handle to operate.
- `lux`: result data (unit:lux)

esp_err_t **light_sensor_acquire_rgbw**(*sensor_light_handle_t sensor*, rgbw_t *rgbw)
Acquire light sensor color result one time. light color includes red green blue and white.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: light sensor handle to operate.
- `rgbw`: result data (unit:lux)

esp_err_t **light_sensor_acquire_uv**(*sensor_light_handle_t sensor*, uv_t *uv)
Acquire light sensor ultra violet result one time. light Ultraviolet includes UVA UVB and UV.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: light sensor handle to operate.
- `uv`: result data (unit:lux)

esp_err_t **light_sensor_sleep**(*sensor_light_handle_t sensor*)
Set sensor to sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: light sensor handle to operate.

esp_err_t **light_sensor_wakeup**(*sensor_light_handle_t sensor*)
Wakeup sensor from sleep mode.

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail
- ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Parameters

- `sensor`: light sensor handle to operate.

esp_err_t **light_sensor_acquire**(*sensor_light_handle_t sensor*, *sensor_data_group_t* **data_group*)
acquire a group of sensor data

Return esp_err_t

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `sensor`: light sensor handle to operate
- `data_group`: acquired data

esp_err_t **light_sensor_control**(*sensor_light_handle_t sensor*, *sensor_command_t cmd*, void **args*)
control sensor mode with control commands and args

Parameters

- `sensor`: light sensor handle to operate
- `cmd`: control commands detailed in sensor_command_t
- `args`: control commands args
  - ESP_OK Success
  - ESP_FAIL Fail
  - ESP_ERR_NOT_SUPPORTED Function not supported on this sensor

Type Definitions

typedef void ***sensor_light_handle_t**
light sensor handle

Enumerations

enum light_sensor_id_t
light sensor id, used for light_sensor_create

Values:

BH1750_ID = 0x01
BH1750 light sensor id

VEML6040_ID
VEML6040 light sensor id

VEML6075_ID
VEML6075 light sensor id

LIGHT_MAX_ID
max light sensor id

8.5 Pressure Sensor

[??]

The pressure sensor can be used to detect the absolute pressure of gases, calculate altitude and etc. It is mainly used in environmental monitoring, altitude measurement and space positioning equipments.

8.5.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	HAL
BME280	Pressure	I2C/SPI	BOSCH	Datesheet	

8.6 Gesture Sensor

[??]

Gesture sensors are generally sensors that convert measurements of reflected infrared light into relevant physical motions and can be used to achieve non-contact interaction between human and machines, etc.

8.6.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	HAL
APDS9960	Light, RGB and Gesture Sensor	I2C	Avago	Datesheet	

CHAPTER

NINE

STORAGE

[??]

9.1 Storage Media

[??]

The supported storage media is listed in the following table:

Name	Key features	Application scenario	Size	Transmission	Speed	Driver	Note
SPI Flash	Can be shared with code, no extra cost	Store parameters, text or images	MB	SPI	40/80 MHz 4-line	SPI Flash Driver	
SD Card	Large capacity, pluggable	Store audio or video files	GB	SDIO/SPI	20/40 MHz 1/4-line	SD/SDIO/MMC Driver	1
eMMC	Large capacity, high-speed read/write	Store audio or video files	GB	SDIO	20/40 MHz 1/4/8-line	SD/SDIO/MMC Driver	2
EEPROM	Can address by byte, low cost	Store parameters	MB	I2C	100 ~ 400 KHz	eeprom	

Note:

1. Only SPI mode is supported for ESP32-S2
2. Not supported for ESP32-S2

9.1.1 SPI Flash

By default, the ESP32/ESP32-S/ESP32-C series chips use NOR flash to store and access users' code and data. The flash can be integrated into the module or chip and is typically 4 MB, 8 MB or 16 MB. For ESP-IDF v4.0 and later versions, the SPI flash component not only supports read and write operations to the main flash, but can also connect to an another external flash for data storage.

Flash can be partitioned using the partition table. Based on functions of the partition table, flash can not only be used to store the binary code generated by users, but can also act as a non-volatile storage (NVS) to store application programming parameters. On top of that, specific flash areas can be mounted to a file system (e.g., FatFS) to store text, images and other files.

The flash chip supports 2-line (DOUT/DIO) and 4-line (QOUT/QIO) operation modes and can be configured to work in 40 MHz or 80 MHz modes. Since the main flash chip can be used directly for data storage without needs for additional memory chips, it is particularly suitable for cost-sensitive applications with small capacity requirements (MB) and high integration needs.

Related documents:

- SPI Flash API

9.1.2 SD Card

The ESP32 supports using either the SDIO or SPI interface to access SD cards. The SDIO interface supports 1/4/8-line modes and supports both the default rate of 20 MHz and the high-speed rate of 40 MHz. Please note that this interface occupies at least 6 GPIOs and only uses fixed pins. The SPI interface can assign any IO for SD cards via the GPIO matrix and supports accessing multiple SD cards via CS pins. In hardware design level, the SPI interface is more flexible for development, but with lower access rate than that of the SDIO interface.

The `SD/SDIO/MMC Driver` in ESP-IDF is wrapped at the protocol layer based on the two access modes of the SD card, and provides the initialization interface and protocol-layer APIs for the SD card. The SD card, with features as large capacity and being pluggable, is widely used in application scenarios with large storage needs such as smart speaker, electronic album and etc.

Related documents:

- SD/SDIO/MMC Driver: supports both SDIO and SPI transmission modes;
- SDMMC Host Driver: supports SDIO mode;
- SD SPI Host Driver: supports SPI mode;
- When using SPI or 1-bit modes, please pay special attention to Pull-up Requirements of Pins.

Examples:

- storage/sd_card: access the SD card which uses FAT file system.

9.1.3 eMMC

The eMMC (embedded MMC) memory chip uses the similar protocol to SD cards and can use the same driver SD/SDIO/MMC Driver as SD cards. However, please note that the eMMC chip can only use SDIO mode and does not support SPI mode. Currently, the eMMC chip supports the default rate of 20 MHz and high-speed rate of 40 MHz in 8-line mode, and supports high-speed rate of 40 MHz in 4-line DDR mode.

The eMMC is generally soldered to the main board as a chip, which is more integrated than SD cards, and is suitable for wearable devices and other scenarios with high storage needs and certain requirements for system integration in the meantime.

Related documents:

- SD/SDIO/MMC Driver;
- Supported eMMC Speed Modes.

9.1.4 EEPROM

EEPROM (e.g., AT24C0X series) is a 1024-16384 bits of serial erasable memory, which can also operate in read-only mode by configuring pin levels. Generally, its storage space is distributed by `word`, with each `word` containing `8-bit` spaces. The EEPROM supports byte addressing and is easy to read and write, making it especially suitable for saving configuration parameters and etc. On top of that, it can also be used in industrial and commercial scenarios with requirements for power consumption and reliability after being optimized.

Adapted EEPROM chips:

Name	Function	Bus	Vendor	Datasheet	Driver
AT24C01/02	1024/2048 bits EEPROM	I2C	Atmel	Datasheet	eeprom

9.2 File System

[??]

Supported file systems:

Key Features	NVS Library	FAT File System	SPIFFS File System
Features	Operates on key-value pairs, with safe interfaces	Operation system supported, strong compatibility	Developed for embedded systems, low resource occupancy
Application Scenarios	Stores parameters	Stores audio, video and other files	Stores audio, video and other files
Size	KB-MB	GB	< 128 MB
Directory Support	X	√	X
Wear Levelling	√	Optional	√
R/W Efficiency	0	0	0
Resources Occupancy	0	0	1
Power Failure Protection	√	X	X
Encryption	√	√	X

Note:

- 0: data not available or not for comparison.
- 1: low RAM occupancy.

9.2.1 NVS Library

Non-volatile storage (NVS) is used to read and write data stored in the flash NVS partition. NVS operated on key-value pairs. Keys are ASCII strings; values can be integers, strings and variable binary large object (BLOB). NVS supports power loss protection and data encryption, and works best for storing many small values, such as application parameters. If you need to store large blobs or strings, please consider using the facilities provided by the FAT file system on top of the wear levelling library.

Related documents:

- Non-volatile storage library.
- For mass production, you can use the NVS Partition Generator Utility.

Examples:

- Write a single integer value: storage/nvs_rw_value.
- Write a blob: storage/nvs_rw_blob.

9.2.2 FAT File System

ESP-IDF uses the FatFs library to work with FAT file system. FatFs is a file system layer independent to platform and storage media that can realize access to physical devices (e.g., flash, SD card) via a unified interface. Although the library can be used directly, many of its features can be accessed via VFS, using the C standard library and POSIX API functions.

The operating system of FAT is compatible with a wide range of mobile storage devices such as USB memory disc or SD cards. And ESP32 series chips can access these common storage devices by supporting the FAT file system.

Related documents:

- Using FatFs with VFS.
- Using FatFs with VFS and SD cards.

Examples:

- storage/sd_card: access the SD card which uses the FAT file system.
- storage/ext_flash_fatfs: access the external flash chip which uses the FAT file system.

9.2.3 SPIFFS File System

SPIFFS is a file system intended for SPI NOR flash devices on embedded targets. It supports wear levelling, file system consistency checks, and more. Users can directly use the Posix interfaces provided by SPIFFS, or use many of its features via VFS.

As a dedicated file system for SPI NOR flash devices on embedded targets, the SPIFFS occupies less RAM resources than FAT and is only used to support flash chips with capacities less than 128 MB.

Related documents:

- SPIFFS Filesystem.
- Two Tools to Generate SPIFFS Images.

Examples:

- storage/spiffs: SPIFFS examples.

9.2.4 Virtual File System (VFS)

The Virtual File System (VFS) component from ESP-IDF provides a unified interface for different file systems (FAT, SPIFFS), and also provides a file-like interface for device drivers.

Related documents:

- Virtual Filesystem Component.

CHAPTER

TEN

MOTOR

[??]

10.1 Servo

[??]

This component uses the LEDC peripheral to generate PWM signals for independent control of servos with up to 16 channels (ESP32 chips support 16 channels and ESP32-S2 chips support 8 channels) at a selectable frequency of 50 ~ 400 Hz. When using this layer of APIs, users only need to specify the servo group, channel and target angle to realize the angle control of a servo.

Generally, there is a reference signal inside the servo generating a fixed period and pulse width, which is used to compare with the input PWM signal to output a voltage difference so as to control the rotation direction and angle of a motor. A common 180 angular rotation servo usually takes 20 ms (50 Hz) as a clock period and 0.5 ~ 2.5 ms as its high level pulse, making it rotates between 0 ~ 180 degrees.

This component can be used in scenarios with lower control accuracy requirements, such as toy cars, remote control robots, home automation, etc.

10.1.1 Instructions

1. Initialization: Use `servo_init()` to initialize a channel. Please note that ESP32 contains two sets of channels as `LEDC_LOW_SPEED_MODE` and `LEDC_HIGH_SPEED_MODE`, while some chip may only support one channel. The configuration items in this step mainly include maximum angle, signal frequency, and minimum and maximum input pulse width to calculate the correspondence between angle and duty cycle; as well as pins and channels to specify the correspondence with chip pins and LEDC channels, respectively;
2. Set a target angle: use `servo_write_angle()` to specify the servo group, channel and target angle so as to realize angle control of the servo;
3. Read the current angle: you can use `servo_read_angle()` to read the current angle of the servo. Please note that this is a theoretical number calculated based on the input signal;
4. De-initialization: you can use `servo_deinit()` to de-initialize a group of channels when a group of servos is used any more.

10.1.2 Application Example

```
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);
```

10.1.3 API Reference

Header File

- 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.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Configure ledc failed

Parameters

- `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

esp_err_t **iot_servo_deinit** (ledc_mode_t *speed_mode*)
Deinitialize ledc for servo.

Return

- ESP_OK Success

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode.

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.

Note This API is not thread-safe

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode.
- `channel`: LEDC channel, select from ledc_channel_t
- `angle`: The angle to go

esp_err_t **iot_servo_read_angle** (ledc_mode_t *speed_mode*, uint8_t *channel*, float *angle)
Read current angle of one channel.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `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

Structures

struct servo_channel_t

Configuration of servo motor channel.

Public Members

gpio_num_t **servo_pin**[**LEDC_CHANNEL_MAX**]

Pin number of pwm output

ledc_channel_t **ch**[**LEDC_CHANNEL_MAX**]

The ledc channel which used

struct servo_config_t

Configuration of servo motor.

Public Members

uint16_t **max_angle**

Servo max angle

uint16_t **min_width_us**

Pulse width corresponding to minimum angle, which is usually 500us

uint16_t **max_width_us**

Pulse width corresponding to maximum angle, which is usually 2500us

uint32_t **freq**

PWM frequency

ledc_timer_t **timer_number**

Timer number of ledc

servo_channel_t **channels**

Channels to use

uint8_t **channel_number**

Total channel number

CHAPTER

ELEVEN

SECURITY & ENCRYPTION

[??]

11.1 Flash ??

11.1.1 ??

- ?? flash encryption ????????????????? SPI flash ??
- flash encryption ?? 256-bit AES key ?? flash ???key ??????? efuse ??????????????????
- ????? flash ?????????????????? boot ????? bootloader ?? flash ??????????????
- ???????????4???????????? flash ???? OTA ?? flash???????????????????? menuconfig??????????????????????????????

11.1.2 ?????

1. make menuconfig ???? “Security features”->”Enable flash encryption on boot”
2. ????????? bootloader, partition table ? app image ????? flash ?
3. ???? boot ? flash ?????????????????? partition????????????????1???? ??????????????????flash????

11.1.3 ????????? boot ?????

1. bootloader ???? efuse ?? FLASH_CRYPT_CNT ?0?????????????????????? key ?? key ????? efuse ???????????????
2. bootloader ?????????? partition ? flash ??????
3. ?????? efuse ?? DISABLE_DL_ENCRYPT, DISABLE_DL_DECRYPT ? DISABLE_DL_CACHE ??????1???? UART bootloader ?????????????? flash ??
4. efuse ?? FLASH_CRYPT_CONFIG ????? 0xf?????????? key ???????????????? flash ??32????????????????? 0xf ??????256?
5. efuse ?? FLASH_CRYPT_CNT ????? 0x01??????? flash ??????????????????“FLASH_CRYPT_CNT”??
6. bootloader ?????????? flash ????? bootloader

11.1.4 □□□□ flash □3□□□□□□

- □□□□ flash □□

  1. make menuconfig □□□ “Security features”->”Enable flash encryption on boot”
  2. □□□□□□□□□□□□□ images □□□ bootloader□□□□ flash □□
  3. □ esp-idf □ components/esptool_py/esptool □□□□□□□ `espefuse.py burn\_efuse FLASH\_CRYPT\_CNT` □□ efuse □□ FLASH_CRYPT_CNT
  4. □□□□□bootloader □□ FLASH_CRYPT_CNT □□□□□□ flash □□□

- □□□□□□□□□□□□□□□□ flash□□□□ esp-idf □ `components/esptool_py/esptool` □□□□□□□ `espefuse.py --port PORT write_protect_efuse FLASH_CRYPT_CNT` □ `FLASH_CRYPT_CNT` □□□□□□□□□□□□□□□□□ bootloader □□□□□ flash □□□□□□

11.1.5 FLASH_CRYPT_CNT

- FLASH_CRYPT_CNT □ flash □□□□□□□□□□□□□□□□□□□ 8-bit □□□□□□□□□□□□ flash □□□□□□□□□□□□□□□□□□□□□□ flash □□□□□□□□
- □ FLASH_CRYPT_CNT □□0,2,4,6,8□□□□□□□1□□bootloader □□ flash □□□□□□□□□□
- □ FLASH_CRYPT_CNT □□1,3,5,7□□□□□□□1□□bootloader □□ flash □□□□□□□□□□□□□□ flash □□□□□□□□□□□
- FLASH_CRYPT_CNT □□□□□□□

  1. □□□□ flash □□□□□□□□0
  2. □□□ flash □□□□□□□□ boot □ bootloader □□□□□□ 0x00□□□□□□ flash □□□□□□□□□□□□□□□□□□□□□□□□ key□□□□□□ flash□□□□□□□□□□□□1□□□□□0x01□
  3. □□ boot □□bootloader □□□□□□□ 0x01□□□□ flash □□□□□□□□□□□□□□□□□□□□
  4. □□□□□□□□□ flash □□□□□□□□□□□□□□ FLASH_CRYPT_CNT□□□□2□ bit □□□ 1□□□□□0x03□
  5. □□□□□□bootloader □□ FLASH_CRYPT_CNT □□□ 0x03□2 bit 1□□□□□□□□□□ flash □□□□□□□□□□ bootloader □ FLASH_CRYPT_CNT □□□0x07□3 bit 1□□flash □□□□□□□
  6. □□□□□□□□□ flash □□□□□□□□□□□□□□ FLASH_CRYPT_CNT□□□□4□ bit □□□ 1□□□□□0x0f□
  7. □□□□□□bootloader □□ FLASH_CRYPT_CNT □□□ 0x0f□4 bit 1□□□□□□□□□□ flash □□□□□□□□□□ bootloader □ FLASH_CRYPT_CNT □□□0x1f□5 bit 1□□flash □□□□□□□
  8. □□□□□□□□□ flash □□□□□□□□□□□□□□ FLASH_CRYPT_CNT□□□□6□ bit □□□ 1□□□□□0x3f□
  9. □□□□□□bootloader □□ FLASH_CRYPT_CNT □□□ 0x4f□6 bit 1□□□□□□□□□□ flash □□□□□□□□□□ bootloader □ FLASH_CRYPT_CNT □□□0x7f□7 bit 1□□flash □□□□□□□
  10. □□□□□□□□□□□□□□□□□ FLASH_CRYPT_CNT□bootloader □□ FLASH_CRYPT_CNT □ 0xff□8 bit 1□□□□□□□□□□□ boot□

11.1.6 ??????

- Bootloader
- Secure boot bootloader digest?? Secure Boot ???? flash ????????? ????"Secure Boot"?"????"???3?
- Partition table
- Partition table ??????? Type ????"app"???
- Partition table ??????? Flags ????"encrypted"???????????????NVS????? flash ????????????????

11.1.7 ??????????????????????

- ????????? flash ?????? flash ??????????????????
  - flash ???????????
  - ??? flash ???????
  - ???? `API esp\_spi\_flash\_mmap()` ??????
  - ? ROM bootloader ?????? bootloader image ??
- ???? API `esp\_partition\_read()` ????????????????? flash ???????????

11.1.8 ??????????????????????????

- ?? API esp_spi_flash_read() ??????
- ROM ????? SPIRead() ??????

11.1.9 ?????????

- ?? API `esp\_partition\_write()` ????????? partition ?????????
- ?? `esp\_spi\_flash\_write()` ???? `write\_encrypted` ?????? true ??????????
- ROM ?? `esp\_rom\_spiflash\_write\_encrypted()` ????????? flash ??? SPIWrite() ????????? flash ?

11.2 ????

11.2.1 ??

- Secure Boot ?????????????????????????????????????? flash ????? partition table ? app images ?????????
- Secure Boot ??? ECDSA ?????? partition table ? app images ?????????ECDSA ?????????/?????????????????????????????????????
- ?? partition table ? app images ???? bootloader ???????????????????? bootloader ?????????Secure Boot ?????? ROM bootloader ?????? bootloader image ????????????? secure boot key ????????????????? efuse ???????????????

11.2.2 ????

- ECDSA ????/???
  - ?? flash ?? PC ???
  - ???????? bootloader image ???? bootloader ?????????????????? flash ?partition table ? app images ???????????????
  - ?????????? partition table ? app images ?????????????????????????????? image ???? boot ???????
- secure bootloader key
  - ???? 256-bit AES key?????? Secure Boot ?????????????? efuse ?????????
  - ??? key ???? bootloader image ??????

11.2.3 ????

1. ?? bootloader image ??? menuconfig ???? secure boot?????? menuconfig ????????/?????????????? bootloader image ??bootloader ??????? secure boot
2. ?? partition table ? app images ????????????????????????????
3. ?????? boot ???? bootloader ????????? secure boot?
   - ??????? secure boot key????? key ??? efuse ??????? key??????? IV ? bootloader image ??? secure digest
   - secure digest ???? IV ??? flash ? 0x0 ????????? boot ??? bootloader image ??????
   - ? menuconfig ??????? JTAG ??? ROM BASIC ???bootloader ?? efuse ????????????????????????????
   - bootloader ???? efuse ?? ABS_DONE_0 ???? secure boot
4. ??????? boot ??ROM bootloader ?? efuse ?? ABS_DONE_0 ???????? flash ??? 0x0 ?????? boot ???? secure digest ???? IV?????? efuse ?? secure boot key ???? IV ???? bootloader image ?????? secure digest??? flash ?? secure digest ???? boot ??????????? bootloader?
5. ?? bootloader ?? bootloader image ???????? flash ?? partition table ? app images ????????????????? boot ? app ???

11.2.4 ????

1. make menuconfig ?? “enable secure boot in bootloader”
2. make menuconfig ???????/???????
3. ????????????? “make” ?????????????????/?????????????????????/?? openssl ?????????????????/????????? openssl?“openssl ecparam -name prime256v1 -genkey -noout -out my_secure_boot_signing_key.pem”??????????/????????????????
4. ???? “make bootloader” ??????? secure boot ? bootloader image
5. ???4????????????? bootloader image ?????????????
6. ???? “make flash” ?????? partition table ? app images
7. ???????? bootloader ??? secure boot ????????? secure boot ??????

11.2.5 ????

- ????????? bootloader image ????????partition table ? app images ???????
- ???????????? secure boot ??????
- ?? OTA ? image ????????????OTA ????????????
- **???????? bootloader??0x1000???????????? 28KB?bootloader)?????? Secure Boot ?????, ????????? Bootloader ?????? ???menuconfig???Bootloader?log???????????????Bootloader???**

11.2.6 ?????? bootloader

- ?????? bootloader image ?????????????????????????? bootloader image ?????????????????? bootloader ????? image ????????? secure boot ??????
- ?????? bootloader ?????secure bootloader key ?? PC ??????? key ????????? key ?????????? key ??digest ? bootloader image ??????????
- ??????
  1. make menuconfig ???? “secure bootloader mode”->”Reflashable”
  2. ?“?????”?????2?3?????????
  3. ????? “make bootloader” ???? 256-bit secure boot key ??
     - ? PC ????? secure boot key ?? efuse ?????
     - ???????????????? secure digest ? bootloader image ???? flash ?
  4. ?“?????”??????6?????

11.2.7 Secure Boot ? Flash Encryption ????

- ???? boot ? secure boot ? flash encrypt ???????????????????? secure boot ????????? flash encrypt ????
- ?? boot ??

11.2.8 ?????????????? flash ? Secure Boot ? Flash encryption

1. make menuconfig ???? secure boot ? flash encrypt?“Secure bootloader mode”??“Reflashable”??????????/??.pem?????
2. ?? bootloader ???? secure boot key?

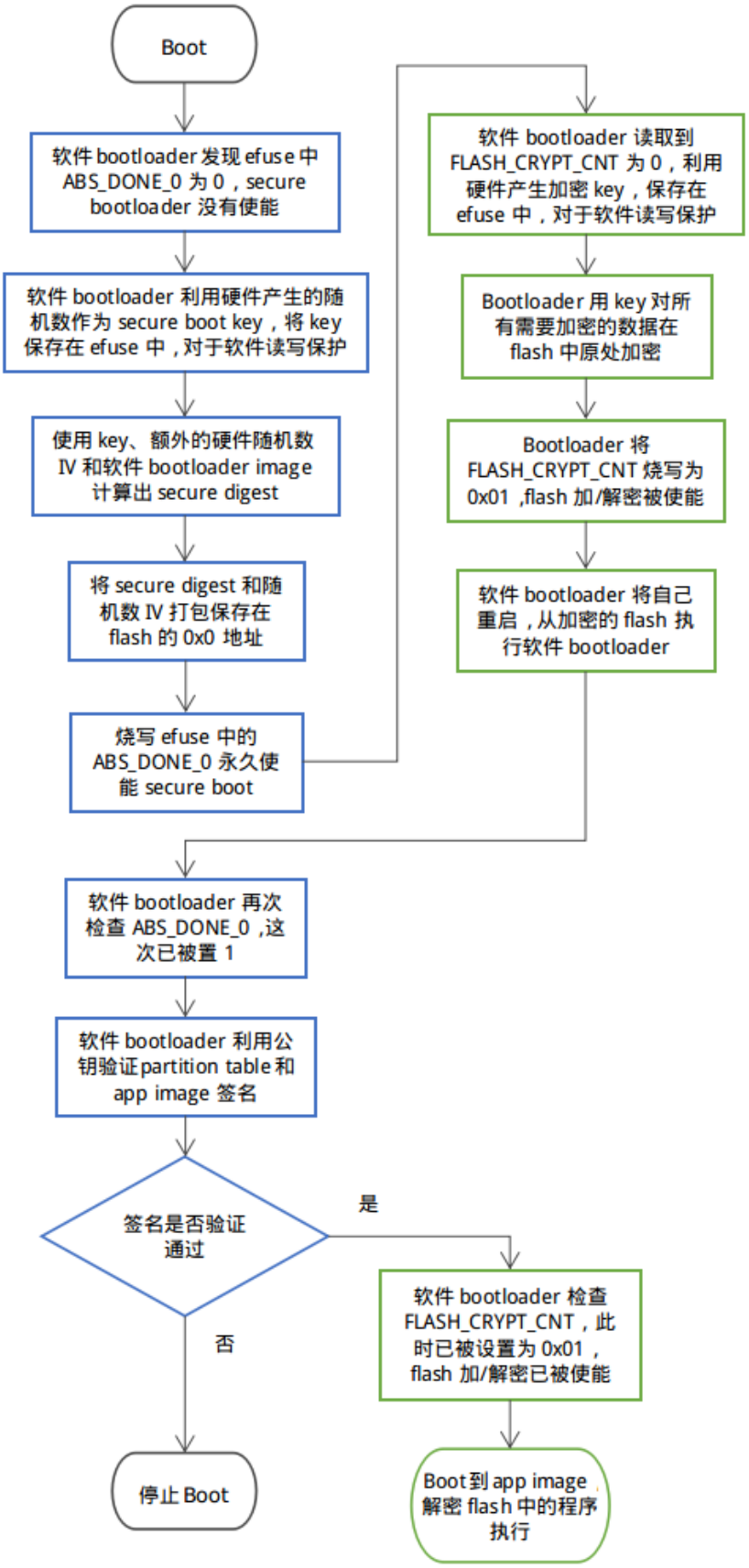
```
make bootloader
```

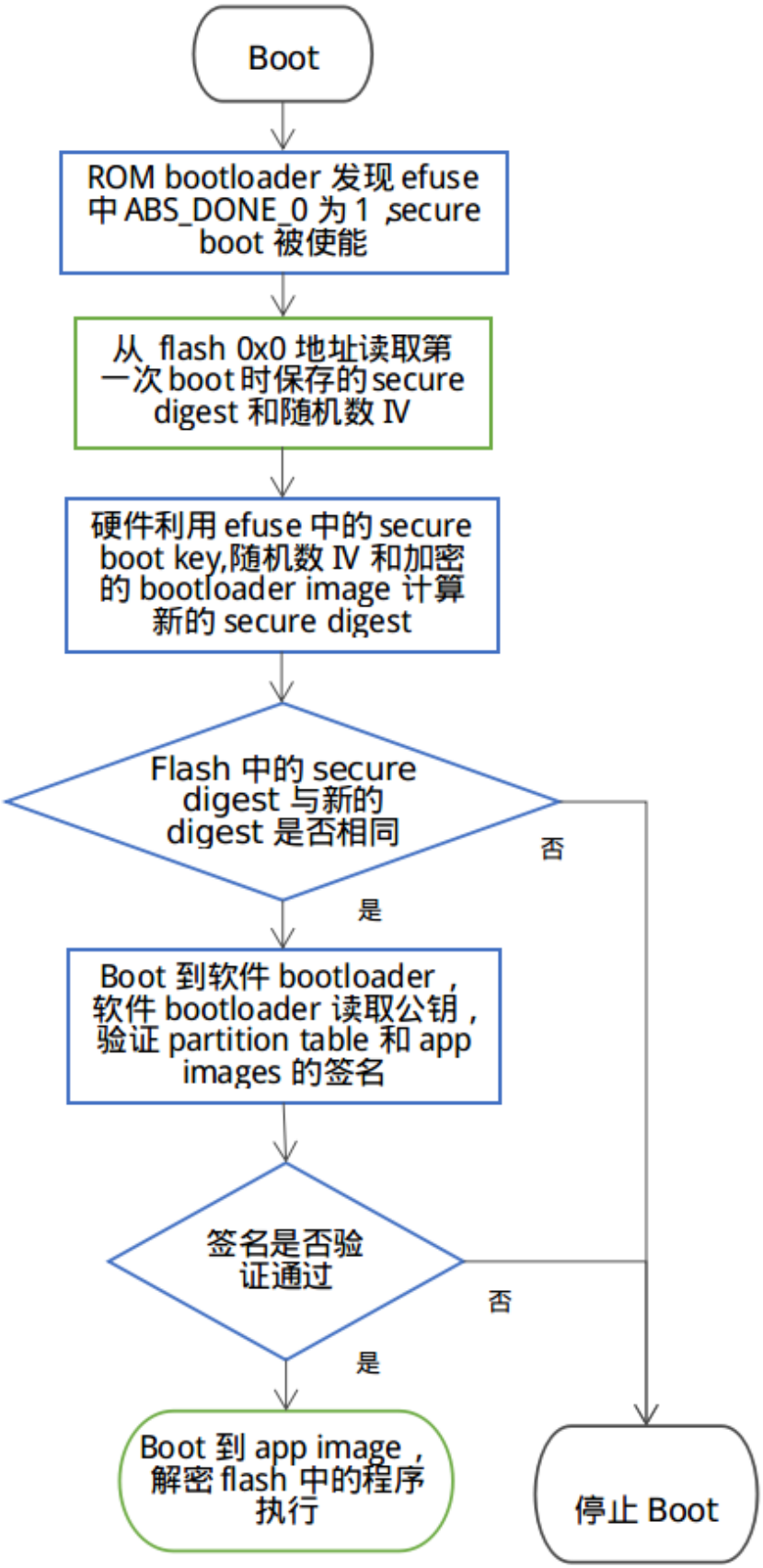
3. ?? key ? bootloader ???? digest ? bootloader

```
python $IDF_PATH/components/esptool_py/esptool/espsecure.py digest_secure_
↪bootloader --keyfile ./build/bootloader/secure_boot_key.bin -o ./build/
↪bootloader/bootloader_with_digest.bin ./build/bootloader/bootloader.bin
```

4. ?? partition_table ? app

```
make partition_table
make app
```


Boot
软件 bootloader 发现 efuse 中 ABS_DONE_0 为 0，secure bootloader 没有使能
软件 bootloader 利用硬件产生的随机数作为 secure boot key，将 key 保存在 efuse 中，对于软件读写保护
使用 key、额外的硬件随机数 IV 和软件 bootloader image 计算出 secure digest
将 secure digest 和随机数 IV 打包保存在 flash 的 0x0 地址
烧写 efuse 中的 ABS_DONE_0 永久使能 secure boot
软件 bootloader 读取到 FLASH_CRYPT_CNT 为 0，利用硬件产生加密 key，保存在 efuse 中，对于软件读写保护
Bootloader 用 key 对所有需要加密的数据在 flash 中原处加密
Bootloader 将 FLASH_CRYPT_CNT 烧写为 0x01 ,flash 加/解密被使能
软件 bootloader 将自己重启，从加密的 flash 执行软件 bootloader
软件 bootloader 再次检查 ABS_DONE_0 ,这次已被置 1
软件 bootloader 利用公钥验证partition table和 app image 签名
签名是否验证通过
是
否
软件 bootloader 检查 FLASH_CRYPT_CNT，此时已被设置为 0x01，flash 加/解密已被使能
停止 Boot
Boot到 app image 解密 flash 中的程序执行


Boot
ROM bootloader 发现 efuse 中 ABS_DONE_0 为 1，secure boot 被使能
从 flash 0x0 地址读取第一次 boot 时保存的 secure digest 和随机数 IV
硬件利用 efuse 中的 secure boot key,随机数 IV 和加密的 bootloader image 计算新的 secure digest
Flash 中的 secure digest 与新的 digest 是否相同
否
是
Boot 到软件 bootloader，软件 bootloader 读取公钥，验证 partition table 和 app images 的签名
签名是否验证通过
否
是
Boot 到 app image，解密 flash 中的程序执行
停止 Boot

5. ???? bin ??

```
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data --
↪keyfile flash_encrypt_key.bin --address 0x0 -o build/bootloader/bootloader_
↪digest_encrypt.bin build/bootloader/bootloader_with_digest.bi
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data --
↪keyfile flash_encrypt_key.bin --address 0x8000 -o build/partitions_singleapp_
↪encrypt.bin build/partitions_singleapp.bin
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data --
↪keyfile flash_encrypt_key.bin --address 0x10000 -o build/iot_encrypt.bin build/
↪iot.bin
```

6. ???????? bin ??

```
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↪flash 0x0 build/bootloader/bootloader_digest_encrypt.bin
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↪flash 0x8000 build/partitions_singleapp_encrypt.bin
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↪flash 0x10000 build/iot_encrypt.bin
```

7. ? flash_encryption_key ?? efuse (?????boot???):

```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_key flash_
↪encryption flash_encrypt_key.bin
```

8. ? secure boot key ??efuse??????boot????:

```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_key secure_boot ./
↪build/bootloader/secure_boot_key.bin
```

9. ?? efuse ??????????boot????

```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse ABS_DONE_0
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse FLASH_CRYPT_
↪CNT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse FLASH_CRYPT_
↪CONFIG 0xf
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_DL_
↪ENCRYPT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_DL_
↪DECRYPT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_DL_
↪CACHE
```

11.3 ??????????

11.3.1 Windows ????

- ????windows??????????????????????????
- ?????????? configure ?????????????? security.conf ??????????????

ITEM	Function	default
debug_enable	????debug????debug??????????pem??????????????????	True
debug_pem_path	??????????????????????????debug?????	
SECURE BOOT		
secure_boot_en	??secure boot??	False
burn_secure_boot_key	??secure boot key??	False
secure_boot_force_write	?????secure boot key block?????key	False
secure_boot_rw_protect	??secure boot key????????	False
FLASH ENCRYPTION		
flash_encryption_en	??flash????	False
burn_flash_encryption_key	??flash encrypt key??	False
flash_encrypt_force_write	?????flash encrypt key block?????key	False
flash_encrypt_rw_protect	??flash encrypt key????????	False
AES KEY	Not used yet	
DISABLE FUNC		
jtag_disable	????JTAG????	False
dl_encrypt_disable	?????????flash????	False
dl_decrypt_disable	?????????flash????	False
dl_cache_disable	?????????flash cache??	False

- ????????????????

11.3.2 ????

????

- ??eptool
  - esptool?????$IDF_PATH/components/esptool_py/esptool/
  - ?????python???

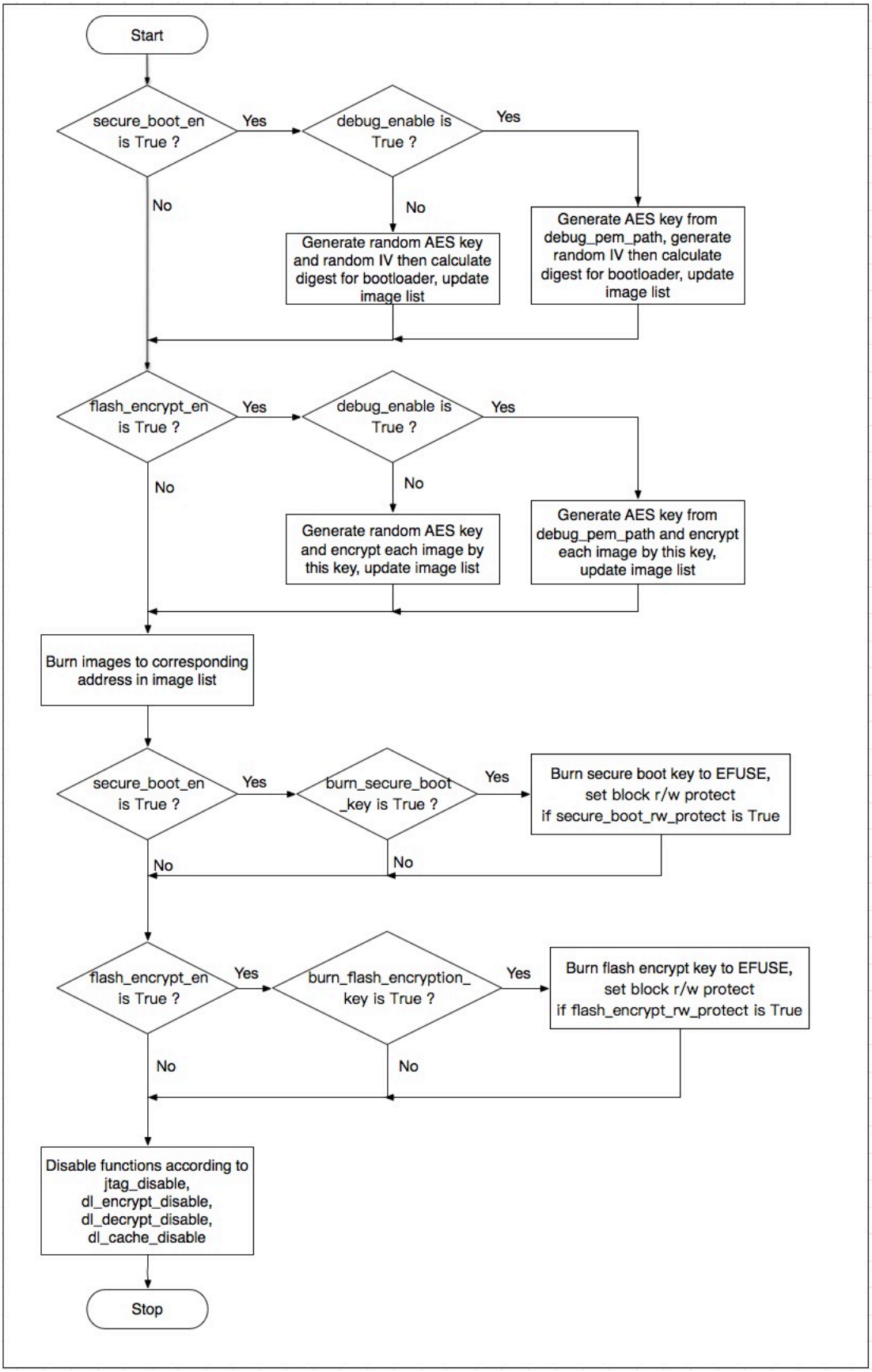
```
pip install esptool
??
pip3 install esptool
```

??1: ?? bootloader ?? security ?????

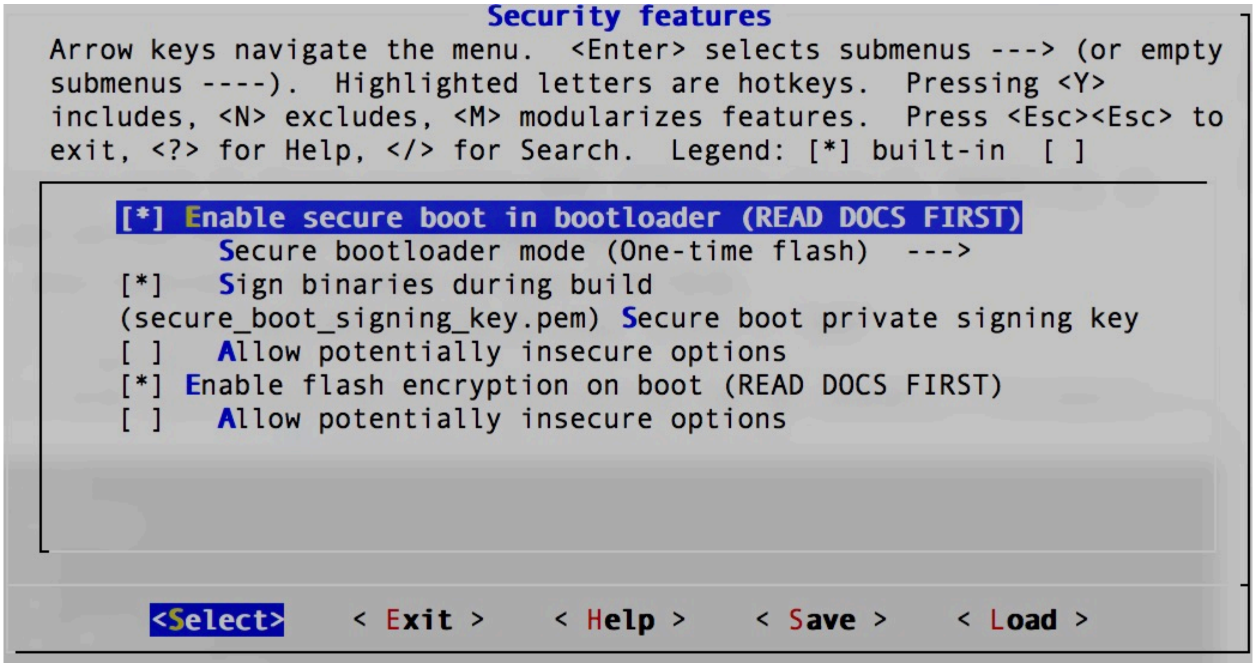
- ??:?????? flash ??????????????????????????????
- ??:??
- ????????? secure boot ? flash encrypton ???????? efuse ?, ????????? efuse block ??????????????????????????
- ?????? images ????????????????? boot ??????
- ?? make menuconfig ?? secure boot ? flash encryption??

1. ????RSA?????

```
espsecure.py generate_signing_key secure_boot_signing_key.pem
or
openssl ecparam -name prime256v1 -genkey -noout -out secure_boot_signing_key.pem
```


Start
secure_boot_en is True ?
Yes
debug_enable is True ?
Yes
No
No
Generate random AES key and random IV then calculate digest for bootloader, update image list
Generate AES key from debug_pem_path, generate random IV then calculate digest for bootloader, update image list
flash_encrypt_en is True ?
Yes
debug_enable is True ?
Yes
No
No
Generate random AES key and encrypt each image by this key, update image list
Generate AES key from debug_pem_path and encrypt each image by this key, update image list
Burn images to corresponding address in image list
secure_boot_en is True ?
Yes
burn_secure_boot _key is True ?
Yes
Burn secure boot key to EFUSE, set block r/w protect if secure_boot_rw_protect is True
No
No
flash_encrypt_en is True ?
Yes
burn_flash_encryption_ key is True ?
Yes
Burn flash encrypt key to EFUSE, set block r/w protect if flash_encrypt_rw_protect is True
No
No
Disable functions according to jtag_disable, dl_encrypt_disable, dl_decrypt_disable, dl_cache_disable
Stop

2. [illegible] menuconfig [illegible] Sign binaries during build,[illegible], [illegible]



3. [illegible]bootloader[illegible]

```
make bootloader
make
```

4. [illegible] esptool [illegible]bin[illegible]flash[illegible],[illegible]example[illegible]hellow-world[illegible]

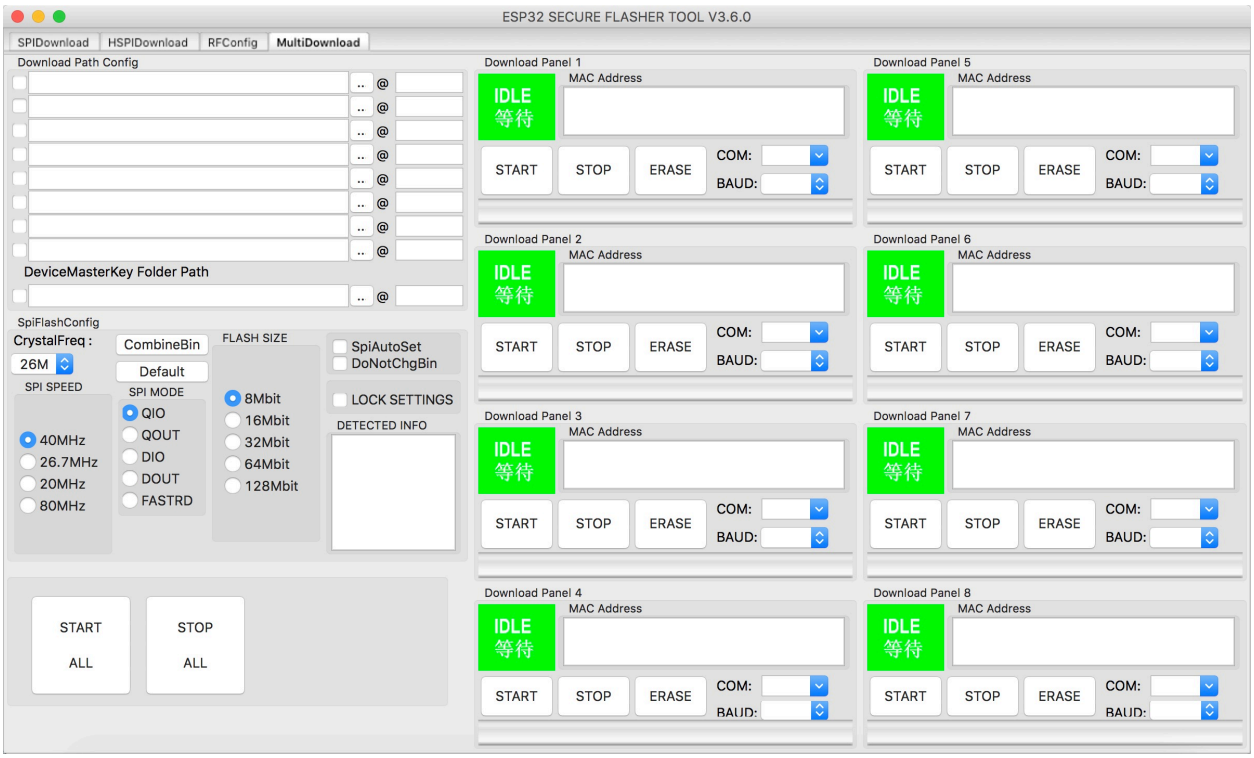
```
bootloader.bin -->   0x1000
partition.bin  -->   0x8000
app.bin        -->  0x10000
python $IDF_PATH/components/esptool_py/esptool/esptool.py --chip esp32 --port /dev/cu.SLAB_USBtoUART --baud 1152000 --before default_reset --after no_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size detect 0x1000 $IDS_PATH/esp-idf/examples/get-started/hello_world/build/bootloader/bootloader.bin 0xf000 $IDF_PATH/esp-idf/examples/get-started/hello_world/build/phy_init_data.bin 0x10000 $IDF_PATH/examples/get-started/hello_world/build/hello-world.bin 0x8000 $IDF_PATH/examples/get-started/hello_world/build/partitions_singleapp.bin
```

Note: [illegible]SPI [illegible]

5. [illegible] window [illegible] security [illegible] security [illegible] bootloader [illegible]

```
[SECURE BOOT]
secure_boot_en * False
[FLASH ENCRYPTION]
flash_encryption_en * False
```

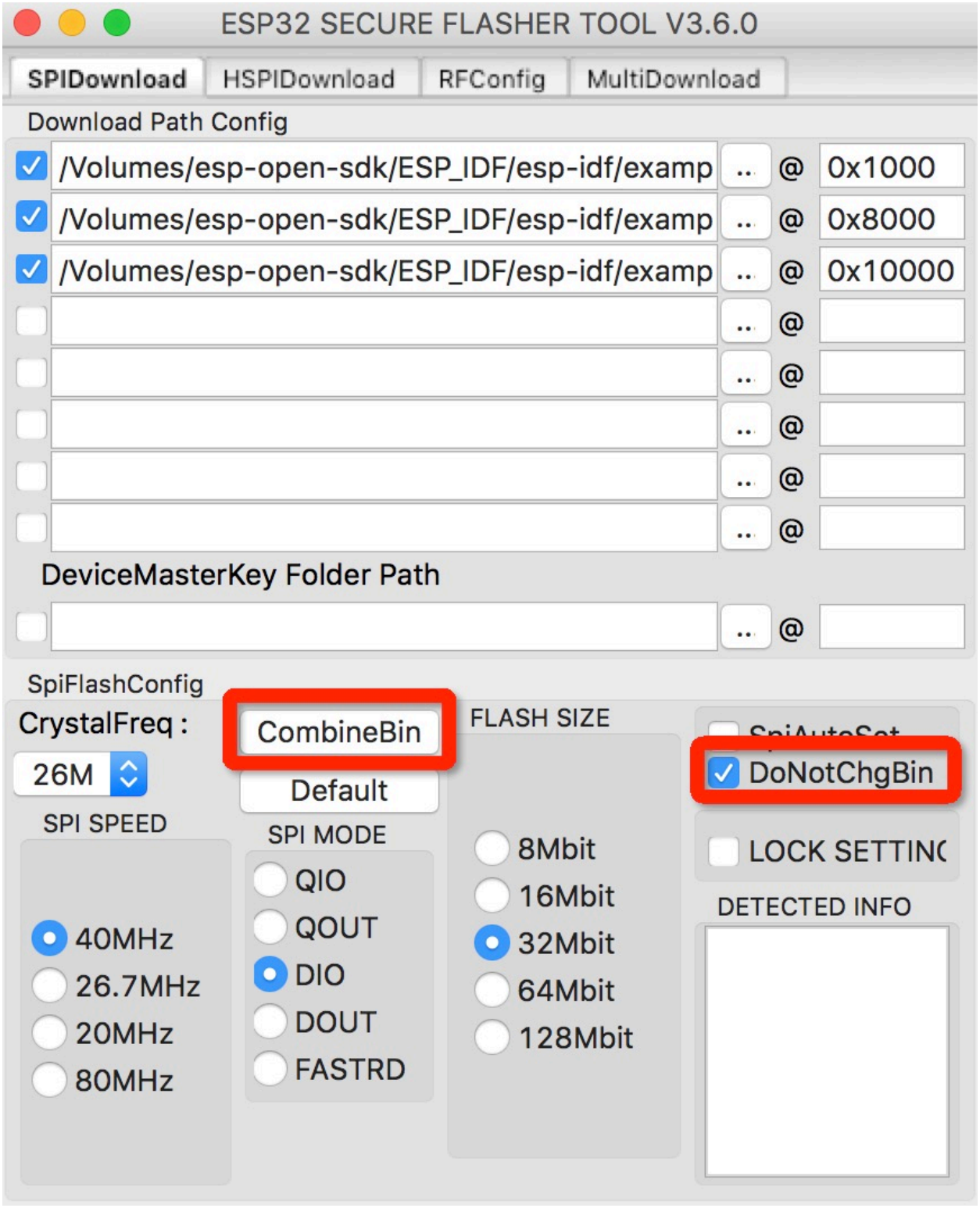
Note: [illegible]



6. ???????????????? combine ??????? bin ???????????????? flash ??????? flash ????????
   - ??bin?????? flash ?????
   - ?? 'DoNotChgBin' ????????????bin??????(SPI??????)?????????
   - ?? 'CombineBin' ????????????bin????
   - ? 'combine' ???????? target.bin??????? Flash ? 0x0 ??????
   - ?????????????????????????? 0xff?????????????
7. ???????????????????? bootloader ?? security ????????????AES???????????EFUSE????????flash????????

Note: ????????????????????????????????????

- ?????
  - ????????????????????app.bin???????????
  - ???????????????????? OTA app ??(????OTA??)?
  - ??????? bootloader ? flash ?????????????????????? partition ???????????
  - ??????? bootloader? flash ???????????
    - ????? secure boot ??????? images ?????? flash ???????? boot ?????? flash
    - ???? secure boot ?????????? flash????????? boot


ESP32 SECURE FLASHER TOOL V3.6.0
SPIDownload
HSPIDownload
RFConfig
MultiDownload
Download Path Config
/Volumes/esp-open-sdk/ESP_IDF/esp-idf/examp
@ 0x1000
/Volumes/esp-open-sdk/ESP_IDF/esp-idf/examp
@ 0x8000
/Volumes/esp-open-sdk/ESP_IDF/esp-idf/examp
@ 0x10000
DeviceMasterKey Folder Path
SpiFlashConfig
CrystalFreq :
26M
CombineBin
Default
FLASH SIZE
DoNotChgBin
LOCK SETTINC
DETECTED INFO
SPI SPEED
40MHz
26.7MHz
20MHz
80MHz
SPI MODE
QIO
QOUT
DIO
DOUT
FASTRD
8Mbit
16Mbit
32Mbit
64Mbit
128Mbit

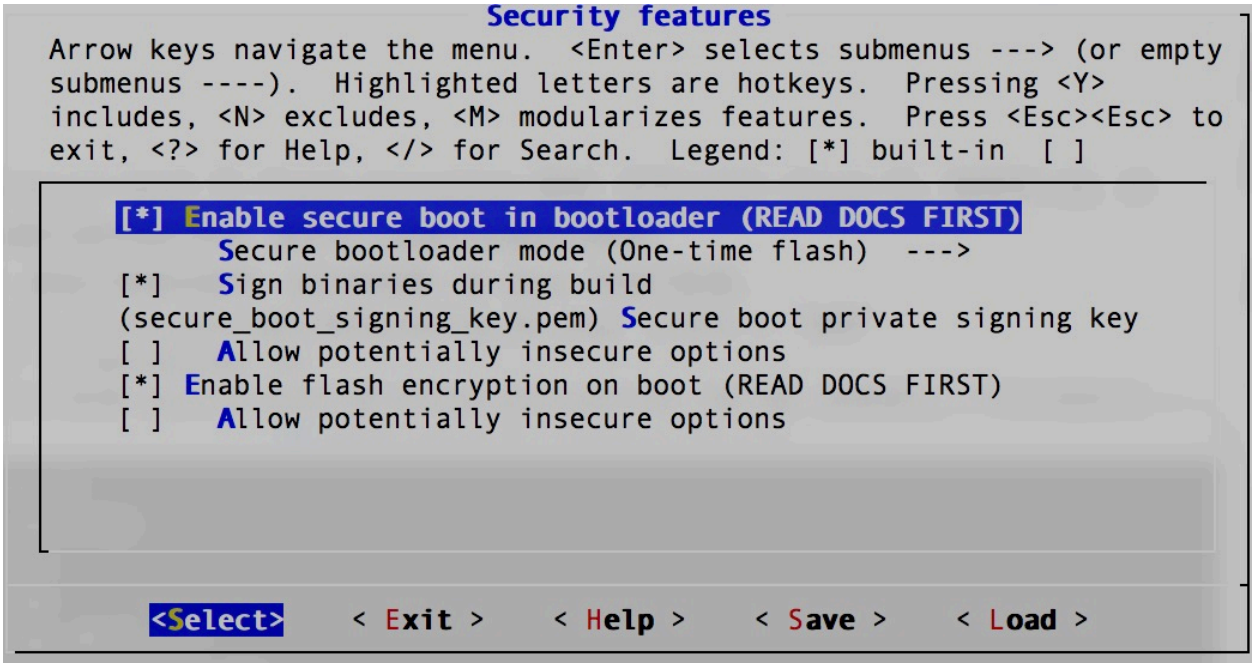
??2: ????????? security ??

- ??: ?????????????? image ????? flash????? efuse. ??????????????????
- ??: ?? flash ??
- ????????? secure boot ? flash encryption????????? make menuconfig ???"enable secure boot in bootloader"??????/???????
- ????????????? secure boot ? flash encryption ??,???????? EFUSE ????
- ?????

1. ????RSA???????????

```
espsecure.py generate_signing_key secure_boot_signing_key.pem
or
openssl ecparam -name prime256v1 -genkey -noout -out secure_boot_signing_key.
↪pem
```

2. ? menuconfig ????Sign binaries during build,???????????, ????



3. ???? bootloader ?????

```
make bootloader
make
```

4. ???????????

```
[DEBUG MODE]
debug_enable * False                    #??debug??????????????pem?????????
debug_pem_path *                        #debug?????????????????????
[SECURE BOOT]
secure_boot_en * True                   #??secure boot??
burn_secure_boot_key * True             #??secure boot key??
secure_boot_force_write * False         #?????secure boot key block?????key
```

(continues on next page)

(continued from previous page)

```
secure_boot_rw_protect * True       #□□secure boot key□□□□□□□
[FLASH ENCRYPTION]
flash_encryption_en * True          #□□flash□□□□
burn_flash_encryption_key * True    #□□flash encrypt key□□
flash_encrypt_force_write * False   #□□□□□flash encrypt key block□□□□□key
flash_encrypt_rw_protect * True     #□□flash encrypt key□□□□□□□
[AES KEY]
aes_key_en * False                  #□□□□□□□□□□□□
burn_aes_key * False                #□□□□□□□□□□□□
[DISABLE FUNC]
jtag_disable * True                 #□□□□JTAG□□□□
dl_encrypt_disable * True           #□□□□□□□□□□flash□□□□
dl_decrypt_disable * True           #□□□□□□□□□□flash□□□□
dl_cache_disable * True             #□□□□□□□□□□□flash cache□□

□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

5. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(□□ flash □□□□□□)□□□□ 'DoNotChgBin' □□□

- □□□□:
  - □□□□□□□□□□□□□□□□□□□□app.bin □□□□□□□□□□
  - □□□□□□□□□□□□□□□□□□□□□□ OTA app □□(□□□ OTA □□)□
  - □□□□□□□□□□ app image □□□□□□□□□□□□□ menuconfig □□ secure boot □□□□□□□□□□□□□□□□□□□□□□ efuse □□ secure boot□□□ app image □□□□□□□□□□□□□□□□

CHAPTER

TWELVE

OTHER RESOURCES

[??]

12.1 GPIO Expander

[??]

With further expansions of the ESP32 chip family, more application scenarios with diverse demands are being introduced, including some that have more requirements on GPIO numbers. The subsequence release of ESP32-S2 and other products have included up to 43 GPIOs, which can greatly alleviate the problem of GPIO resource constraint. If this still could not meet your demand, you can also add GPIO expansion chips to ESP32 to have more GPIO resources, such as using the I2C-based GPIO expansion module MCP23017, which can expand 16 GPIO ports per module and mount up to 8 expansion modules simultaneously thus expanding additional 128 GPIO ports totally.

12.1.1 Adapted Products

Name	Function	Bus	Vendor	Datasheet	Driver
MCP23017	16-bit I/O expander	I2C	Microchip	Datasheet	mcp23017

12.2 ADC Range Extension Solution

[??]

12.2.1 ESP32-S3 ADC Range Extension

The maximum effective range of the ESP32-S3 ADC is 0 ~ 3100 mV. Through the external voltage divider circuit, it can meet most of the functions such as the ADC button or battery voltage detection. However, for applications such as NTC (Negative Temperature Coefficient) based temperature measurement, it may need to support full-scale (0 ~ 3300 mV) measurement. ESP32-S3 can adjust the ADC offset through registers, and combined with the nonlinear compensation method of the high voltage area, the expansion of the ADC range can be implemented.

The process is as follows:

1. Measure the first voltage value using the default offset
2. If the measured voltage is less than 2900 mV, the first voltage is directly output as the measurement result

3. Else if the measured voltage is greater than 2900 mV, increase the offset value to take the secondary measurement. Then carried out the nonlinear correction calculation on the secondary value, will be output as the final measurement result.
4. Restore the offset value once measurement is completed

Overall, during each ADC measurement, there will be 1-2 times ADC reading. For most application scenarios, the measurement delay introduced by this scheme is negligible.

12.2.2 Patch Use Guide

At present, the patch file is developed based on ESP-IDF `release/v4.4` branch:

1. Please make sure ESP-IDF has been `checked out` to the `release/v4.4` branch
2. Please download file `esp32s3_adc_range_to_3100.patch` to anywhere you want
3. Using command `git am --signoff < esp32s3_adc_range_to_3100.patch` to apply the patch to ESP-IDF

12.2.3 API Guide

1. To get the range expansion result, users must directly use `esp_adc_cal_get_voltage` to get the voltage of `ADC1` or `ADC2`.
2. Other APIs of ESP-IDF ADC are not affected, and the read results are consistent with the default results

CHAPTER

THIRTEEN

CONTRIBUTIONS GUIDE

We welcome contributions to the esp-iot-solution project!

13.1 How to Contribute

Contributions to esp-iot-solution - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via Github Pull Requests.

13.2 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* ?
- 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.

13.3 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.

13.4 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.

13.5 Related Documents

13.5.1 esp-iot-solution ????

????

- ??????????
- ??????????
- ??????????
- ?? ESP-IDF ????
- ??????????????

????

- components??????????????????????????????????README ????????
- docs?rst ????????????????????API ???
- examples??????????????????
- tools?CI ?????????

???

- ???? .c ???????? .h ???
- ????????? .h ?????? .h????????????????
- ??????????????????????
- ???????????????????
- ??????????????????????????;
- ??????????????????????????????????????

```
#ifndef _IOT_I2C_BUS_H_
#define _IOT_I2C_BUS_H_

#endif
```

- ??????? extern “C” ?????? C/C++ ??????

```
#ifdef __cplusplus
extern "C"
{
#endif

//c code

#ifdef __cplusplus
}
#endif
```

??

- ?? VSCODE ?? Doxygen Documentation Generator ??????????
- ??????????????
- ??

????????????

```
/**
* @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);
```

????????????

```
/**
* @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);
```

- ??????????????????????????

```
// Copyright 2019-2020 Espressif Systems (Shanghai) PTE LTD
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at

//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
```

(continues on next page)

(continued from previous page)

```
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
```

????

- ??????????????????
- ?????????????????????? `static`?
- ????????????????????????????????????
- ??????????????????????
- ????????????????????????
- ??????????????????????
- ????????`snake_case`?????????????????? `_` ;
- ????????????????????????????????

??????	?????	??
iot_type_xxx	iot_sensor_xxx; iot_board_xxx; iot_storage_…	????? iot ??
type_xxx	imu_xxx; light_xxx; eeprom_xxx	?????????
name_xxx	mpu6050_xxx;	?? driver??????????????????
xxx_creat / xxx_delete		?????
xxx_read / xxx_write		????

????

- ?????????????????????? `get_` `set_` ?????????
- ?????????????????????? `static`?
- ????????? `g_` ?????????????? `s_` ???
- ??????????????????????
- ????????????????
- ??????????????????????
- ?????????????????????? NULL;
- ???????`snake_case`?????????????????? `_` ;
- ???????????? `data` ????? `dat`?
- ????????????????????????????????????
- ????????

??	??	??
????	????	x
???????	static ?? ? g_ ???????	static uint32_t g_connect_num = 0;
???????	static ?? ? s_ ???????	static uint32_t s_connect_num= 0;
???????	????? `i` `j` `k`	
????	abbreviations-in-code	addr,buf ,cfg , cmd, , ctrl,

- ???????

??	??	??	??	??	??	??	??
addr	address	id	identifier	len	length	ptr	pointer
buf	buffer	info	information	obj	object	ret	return
cfg	command	hdr	header	param	parameter	temp	temporary?temperature
cmd	command	init	initialize	pos	position	ts	timestamp

????

- ???snake_case??? `_t` ??

```
typedef int signed_32_bit_t;
```

- ????? typedef ??????????

```
typedef enum {
    MODULE_FOO_ONE,
    MODULE_FOO_TWO,
    MODULE_FOO_THREE
} module_foo_t;
```

????????

????? ESP-IDF ??

1. ??

???????? **4 ???**????????????????????????????? tab ???? 4 ????

2. ????

????????????????????????????

```
void function1()
{
    do_one_thing();
    do_another_thing();
                                // INCORRECT, don't place empty line here
}
                                // place empty line here
```

(continues on next page)

(continued from previous page)

```
void function2()
{
                                // INCORRECT, don't use an empty line here
    int var = 0;
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
    }
}
```

[illegible] 120 [illegible]

3. [illegible]

[illegible]

```
if (condition) {    // correct
    // ...
}

switch (n) {        // correct
    case 0:
        // ...
}

for(int i = 0; i < CONST; ++i) {    // INCORRECT
    // ...
}
```

[illegible]

```
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
```

. [illegible] -> [illegible]

[illegible]:

```
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);
```

- [illegible](PIN_CAM_VSYNC)[illegible]
- [illegible]

4. ??

??????????????????

```
// This is correct:
void function(int arg)
{

}

// NOT like this:
void function(int arg) {

}
```

??????????????????????????????

```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

5. ??

// ?????????????????????????? //? / * * / ????

??????????????????????????????????????

- ??????????????????

```
void init_something()
{
    setup_dma();
    // load_resources();                // WHY is this thing commented, asks the␣
↪reader?
    start_timer();
}
```

- ????????????????????????????????????? git ???

```
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` ??? `#if 0 ... #endif` ??????????????????

- ????????????????????????????????? git ??

```
void init_something()
{
```

(continues on next page)

(continued from previous page)

```
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
```

6. [illegible]

commit [illegible] LF[illegible]Unix[illegible]

Windows [illegible] git [illegible] checkout [illegible] CRLF[illegible]Windows [illegible] core.autocrlf [illegible] commit [illegible] LF [illegible] Github [illegible] [illegible] MSYS2 [illegible] Unix [illegible] ESP-IDF [illegible] LF[illegible]Unix [illegible]

[illegible] commit [illegible] LF [illegible] MSYS2 [illegible] Unix [illegible] Unix[illegible] IDF [illegible] checkout [illegible]

```
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs dos2unix &
↪& git commit -a --amend --no-edit --allow-empty' master
```

([illegible] master [illegible])

[illegible]

```
dos2unix FILENAME
```

[illegible]

```
git commit --amend
```

7. [illegible]

[illegible] astyle [illegible]

[illegible]

[illegible]

```
tools/format.sh components/my_component/file.c
```

CMake ????

- ??? 4 ???
- ????? 120 ???? ???/?????
- ??? endforeach()?endif() ??????????????????
- ???????????????(with_underscores)?
- ???????????????????(with_underscores)?
- ???????????????????(WITH_UNDERSCORES)?
- ?????? cmake-lint ?????????

INDEX

Symbols

B

C

D

F

G

H

I

K

L

M

N

P

R

S

T

U

V