Button function

1. Experimental purpose

Read the button status on the microROS control board and print the free memory capacity of the ESP32S3 chipset.

2. Hardware connection

As shown in the figure below, the buttons are onboard components, so there is no need to connect other external devices. You only need to connect the type-C data cable to the computer and the microROS control board for the firmware burning function.
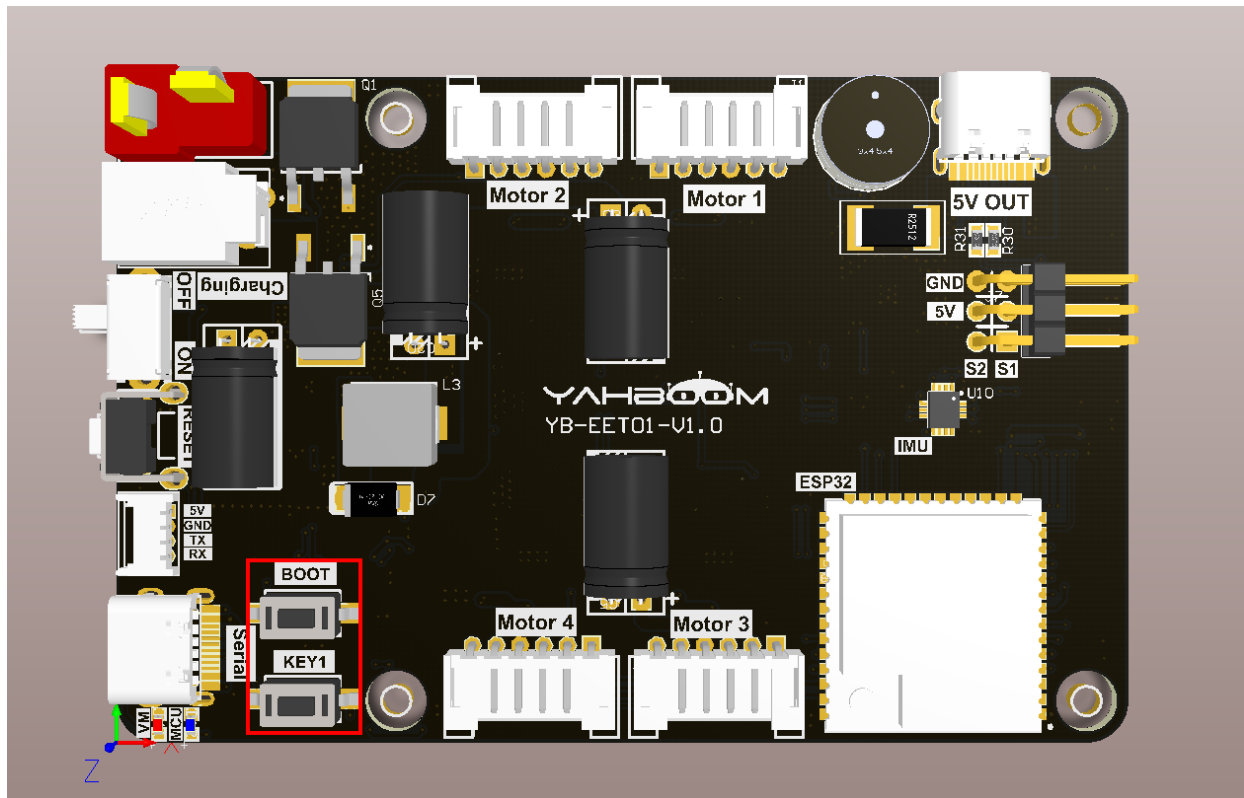
The microROS control board has two boot buttons and a custom function button. The boot button and the reset button can allow the ESP32 to enter the burning mode manually. After the boot is completed, the boot button can also be used as a custom function button.

3. Core code analysis

The virtual machine path corresponding to the program source code is:

```
~/esp/Samples/esp32_samples/key
```

Initialize the KEY peripherals, where KEY_GPIO_BOOT0 corresponds to GPIO0 of the hardware circuit, KEY_GPIO_USER1 corresponds to GPIO42 of the hardware circuit, and the GPIO mode is set to pull-up input mode.

```
void Key_Init(void)
{
    // zero-initialize the config structure.
    gpio_config_t io_conf = {};
    //disable interrupt 禁用中断
    io_conf.intr_type = GPIO_INTR_DISABLE;
    //set as input mode 设置为输入模式
    io_conf.mode = GPIO_MODE_INPUT;
    //bit mask of the pins that you want to set 引脚编号设置
    io_conf.pin_bit_mask = ((1ULL<<KEY_GPIO_BOOT0)|(1ULL<<KEY_GPIO_USER1));
    //disable pull-down mode 禁用下拉
    io_conf.pull_down_en = 0;
    //enable pull-up mode 使能上拉
    io_conf.pull_up_en = 1;
    //configure GPIO with the given settings 配置GPIO口
    gpio_config(&io_conf);
}
```

Since the functional logic of the boot button Key0 and the custom button Key1 are the same, here we take the button Key1 as an example to analyze the key pressing process.

Determine whether key1 is pressed. If it is pressed, it returns KEY_STATE_PRESS. If it is released, it returns KEY_STATE_RELEASE.

```
static uint8_t Key1_is_Pressed(void)
{
    uint8_t key_state = KEY_STATE_RELEASE;
    if (!gpio_get_level(KEY_GPIO_USER1))
    {
        key_state = KEY_STATE_PRESS;
    }
    return key_state;
}
```

Non-blocking reading of the status of key1 (software anti-shake) needs to be called every 10 milliseconds, and KEY_STATE_PRESS is returned every time the key is pressed.

```
uint8_t Key1_Read_State(void)
{
```

```
    static uint16_t key_state = 0;

    if (Key1_is_Pressed() == KEY_STATE_PRESS)
    {
        if (key_state < 3)
        {
            key_state++;
        }
    }
    else
    {
        key_state = 0;
    }
    if (key_state == 2)
    {
        return KEY_STATE_PRESS;
    }
    return KEY_STATE_RELEASE;
}
```

Call the Key_Init function in app_main to initialize the keys, and read the status of Key0 and Key1 in the loop (every 10 milliseconds). When the keys are pressed, a key press prompt is printed, and the current free memory capacity is printed.

```
Key_Init();

while (1)
{
    if (Key0_Read_State() == KEY_STATE_PRESS)
    {
        ESP_LOGI(TAG, "KEY 0 PRESS");
        ESP_LOGI(TAG, "free internal heap size = %ld",
esp_get_free_internal_heap_size());
    }
    if (Key1_Read_State() == KEY_STATE_PRESS)
    {
        ESP_LOGI(TAG, "KEY 1 PRESS");
        printf("free heap size = %ld\r\n", esp_get_free_heap_size());
    }

    vTaskDelay(pdMS_TO_TICKS(10));
}
```

4. Compile and download the burning firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

Enter the project directory

```
cd ~/esp/Samples/esp32_samples/key
```

Compile, burn, and open the serial port simulator

```
idf.py build flash monitor
```

If you need to exit the serial port simulator, press Ctrl+].

5. Experimental results

The serial port simulator prints the "hello yahboom" welcome message. When we press the boot button Key0, the serial port simulator prints "KEY 0 PRESS" and displays the current internal free memory capacity. When we press the custom key Key1, the serial port simulator will print "KEY 1 PRESS" and display the current free memory capacity.

```
I (313) main_task: Calling app_main()
hello yahboom
I (318) KEY_MAIN: Nice to meet you!
I (322) gpio: GPIO[0]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0|
Intr:0
I (331) gpio: GPIO[42]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0|
 Intr:0
I (5161) KEY_MAIN: KEY 0 PRESS
I (5161) KEY_MAIN: free internal heap size = 382236
I (6271) KEY_MAIN: KEY 1 PRESS
free heap size = 390008
```