3.2 Light up RGB

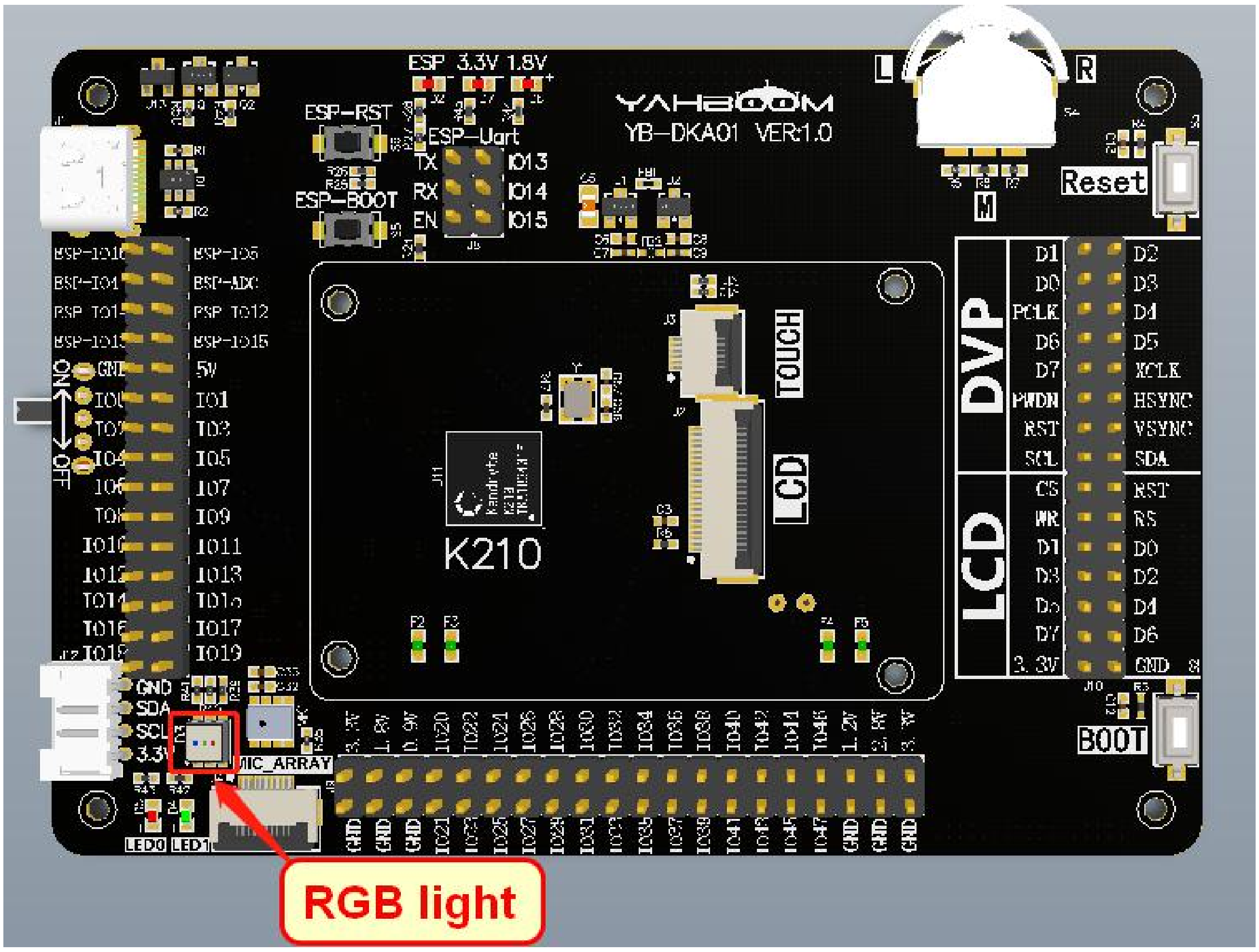
1. Experiment purpose

In this lesson mainly learns the high-speed GPIOHS of K210 and light up the RGB lights.

2.Experiment preparation

2.1 components

RGB light, as shown below.

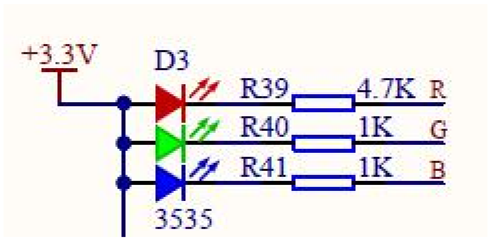


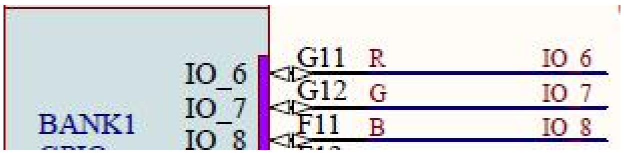
2.2 Component characteristics

RGB lights can light up red, green, and blue lights. Then, it can be combined into other colors according to different brightness. Such as yellow, purple, etc.

2.3 Hardware connection

RGB light have been soldered on the K210 development board by default. R pin is connected to IO6, G pin is connected to IO7, B pin is connected to IO8.





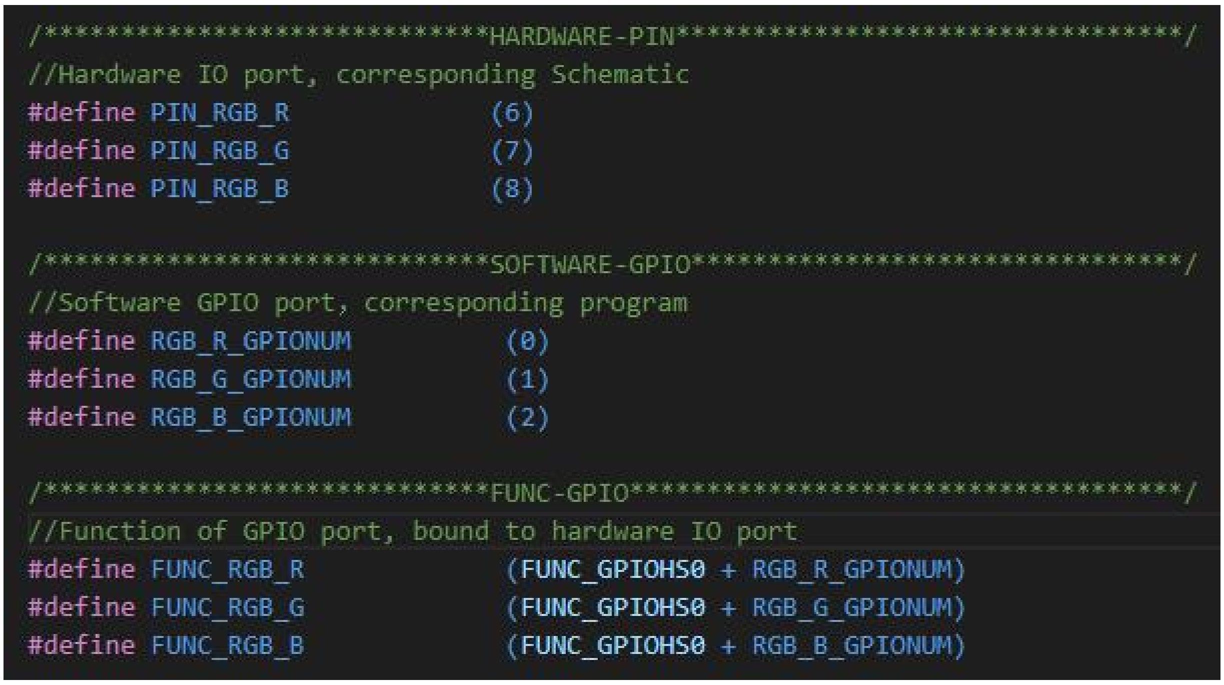
2.4 SDK API function
Header file is gpio.h
There are 32 High speed GPIOHS. They can be configured to input and output mode, can be configured to pull-up, pull-down or high-impedance. Each IO with a separate interrupt source, interrupt supports edge and level trigger. Each IO can be assigned to one of the 48 pins on FPIOA.

We will provide the following ports for users.

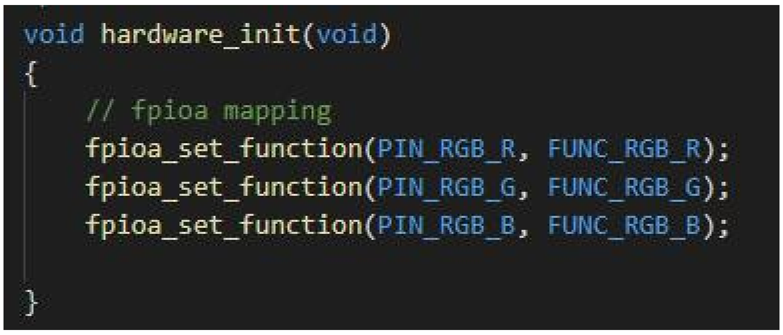
- gpiohs_set_drive_mode: set GPIO port input or output mode
- gpiohs_set_pin: set GPIO pin level high/low
- gpiohs_get_pin: read GPIO pin level
- gpiohs_set_pin_edge: set the interrupt trigger level
- gpiohs_set_irq (Versions after 0.6.0 are no longer supported, please use gpiohs_irq_register)
- gpiohs_irq_register: register pin interrupt service
- gpiohs_irq_unregister: unregister pin interrupt service

3. Experimental principle
The RGB light is composed of three LEDs with red, green and blue colors, so the light-emitting principle is the same as that of LED light. For RGB light, the three LED lights are close together so that different colors can be set to achieve different colors.

4. Experiment procedure
4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.
!Note: All the operations in the program are software pins, so you need to map the hardware pins to software GPIO functions. Then, you can directly operate the software GPIO.

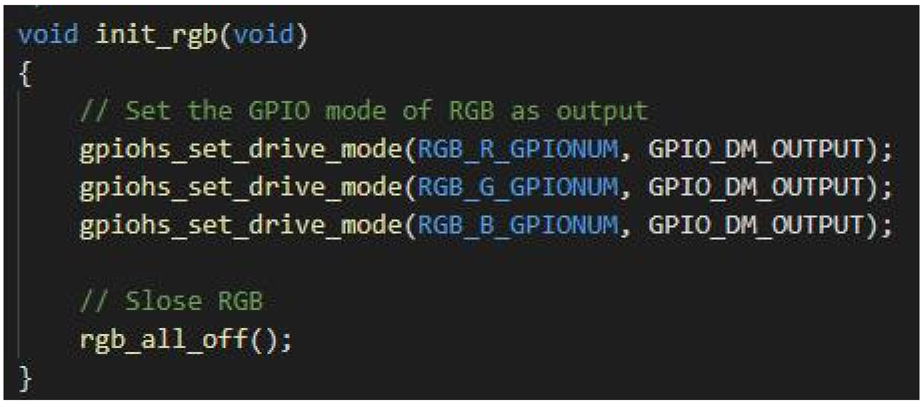


```
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_RGB_R            (6)
#define PIN_RGB_G            (7)
#define PIN_RGB_B            (8)

/*****************************SOFTWARE-GPIO********************************/
//Software GPIO port, corresponding program
#define RGB_R_GPIONUM         (0)
#define RGB_G_GPIONUM         (1)
#define RGB_B_GPIONUM         (2)

/*****************************FUNC-GPIO************************************/
//Function of GPIO port, bound to hardware IO port
#define FUNC_RGB_R            (FUNC_GPIOHS0 + RGB_R_GPIONUM)
#define FUNC_RGB_G            (FUNC_GPIOHS0 + RGB_G_GPIONUM)
#define FUNC_RGB_B            (FUNC_GPIOHS0 + RGB_B_GPIONUM)
```



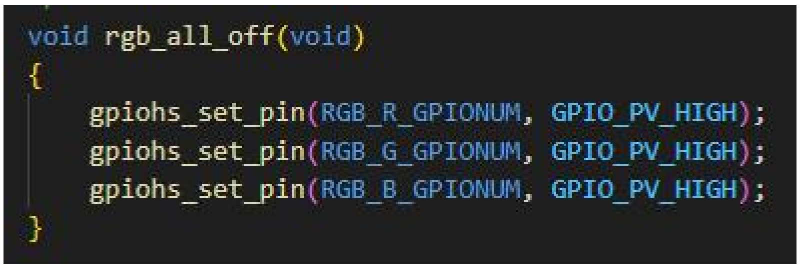
```
void hardware_init(void)
{
    // fpioa mapping
    fpioa_set_function(PIN_RGB_R, FUNC_RGB_R);
    fpioa_set_function(PIN_RGB_G, FUNC_RGB_G);
    fpioa_set_function(PIN_RGB_B, FUNC_RGB_B);

}
```

4.2 We need to initialize the pin before using the RGB light, that is, set the software GPIO of the RGB light to the output mode.



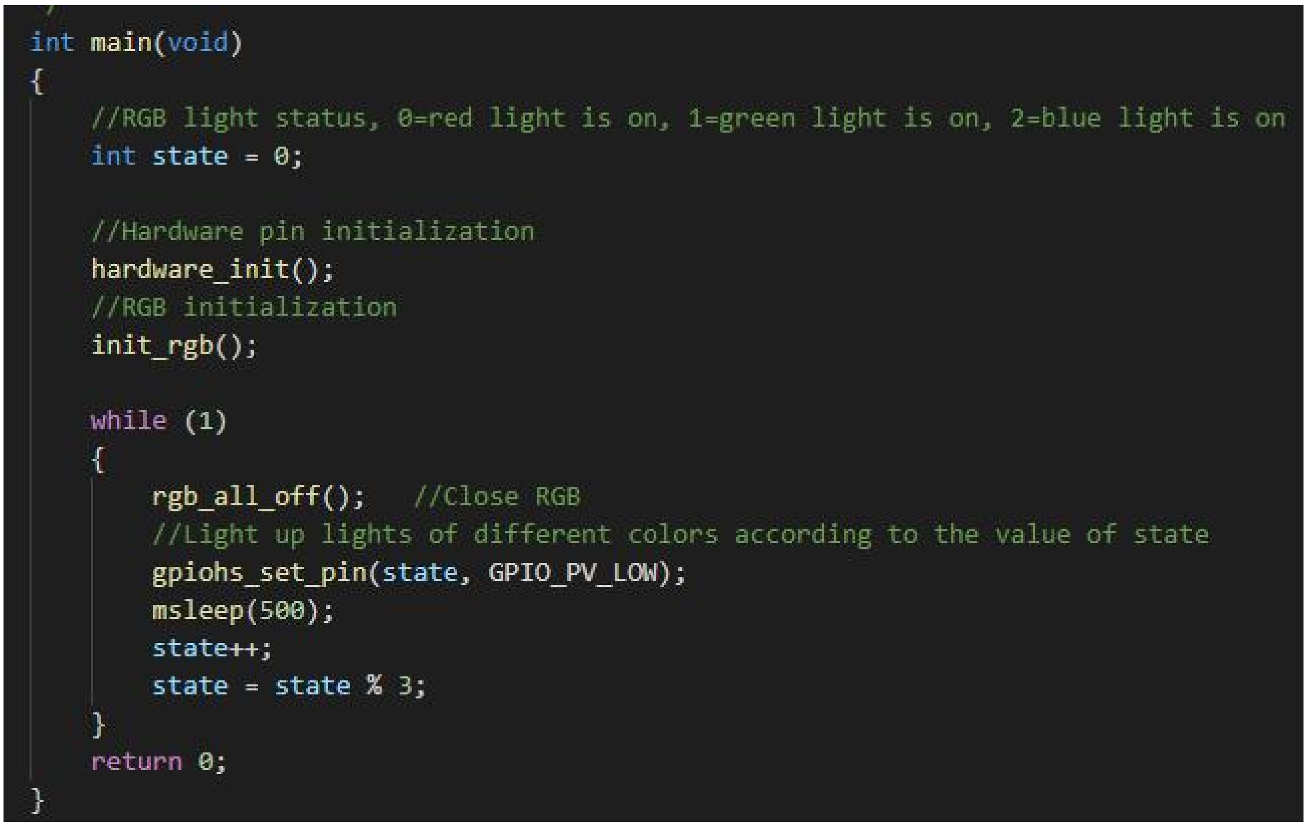
```
void init_rgb(void)
{
    // Set the GPIO mode of RGB as output
    gpiohs_set_drive_mode(RGB_R_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_G_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_B_GPIONUM, GPIO_DM_OUTPUT);

    // Slose RGB
    rgb_all_off();
}
```

4.3 Then, set the GPIO of the RGB light to high level to turn off the RGB light.



```c
void rgb_all_off(void)
{
    gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
}
```

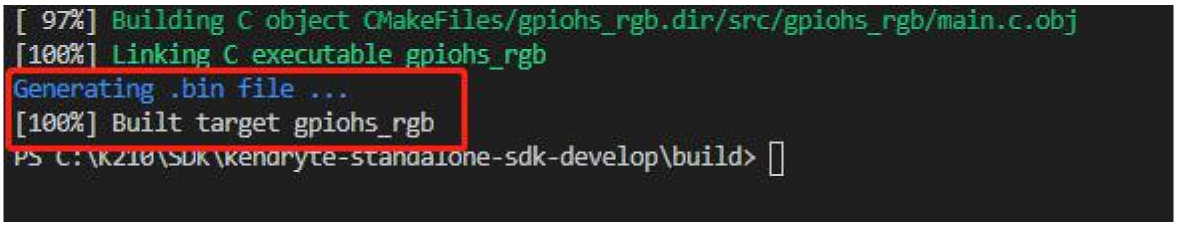
4.4 Finally, in the while loop, modify the value of state every 0.5 seconds to change the color of the RGB light.
Since we didn't know which light was on last time, we need to turn off the RGB light every time before setting the color of the light.
"state = state% 3"; The function of this statement is to limit the value of state to 0,1,2, which corresponds to the software GPIO.



```c
int main(void)
{
    //RGB light status, 0=red light is on, 1=green light is on, 2=blue light is on
    int state = 0;

    //Hardware pin initialization
    hardware_init();
    //RGB initialization
    init_rgb();

    while (1)
    {
        rgb_all_off();    //Close RGB
        //Light up lights of different colors according to the value of state
        gpiohs_set_pin(state, GPIO_PV_LOW);
        msleep(500);
        state++;
        state = state % 3;
    }
    return 0;
}
```

4.5 Compile and debug, burn and run
Copy the gpio_rgb folder to the src directory in the SDK.
Then, enter the build directory and run the following command to compile.

```
cmake .. -DPROJ=gpiohs_rgb -G "MinGW Makefiles"
make
```

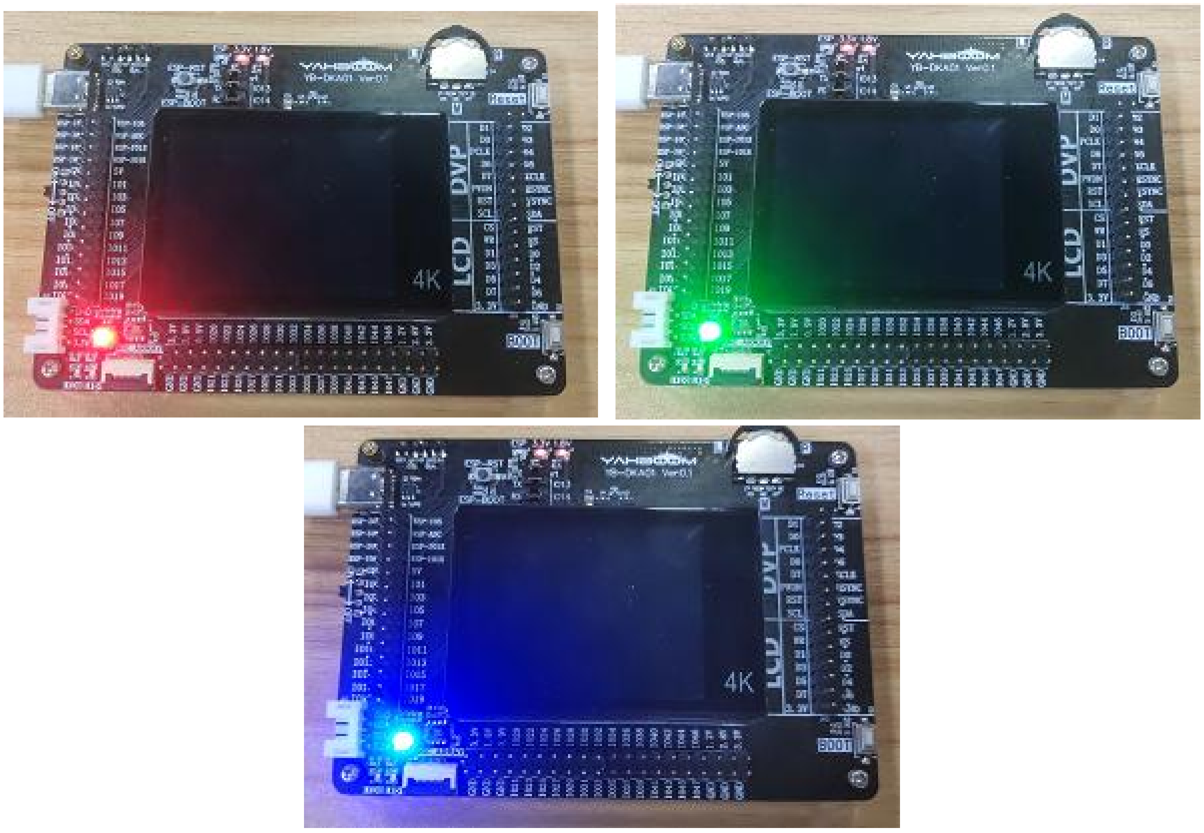


```
[ 97%] Building C object CMakeFiles/gpiohs_rgb.dir/src/gpiohs_rgb/main.c.obj
[100%] Linking C executable gpiohs_rgb
Generating .bin file ...
[100%] Built target gpiohs_rgb
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build> 
```

After the compilation is complete, the **gpio_rgb.bin** file will be generated in the build folder.
We need to use the type-C data cable to connect the computer and the K210 development board.
Open kflash, select the corresponding device, and then burn the **gpio_rgb.bin** file to the K210 development board.

5. Experimental phenomenon

RGB light will light up in red for 0.5 --> light up in green for 0.5s --> light up in blue for 0.5s. Keep the loop in this state.



6. Experiment summary

6.1 The lighting principle of RGB lights and LED lights is the same.
6.2 Each IO of high-speed gpiohs possess a separate interrupt source, but general-purpose gpio shares a total interrupt source.

Appendix -- API

Header file is gpio.h

1) gpiohs_set_drive_mode

Function: set GPIOHS drive mode.

Function prototype: **void gpiohs_set_drive_mode(uint8_t pin, gpio_drive_mode_t mode)**

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input
mode	GPIO drive mode	Input

Return value: No

2) gpiohs_set_pin

Function: set GPIOHS value.

Function prototype: **void gpiohs_set_pin(uint8_t pin, gpio_pin_value_t value)**

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input
value	GPIO value	Input

Return value: No

3) gpiohs_get_pin

Function: Get the GPIOHS pin value.

Function prototype: **gpiohs_pin_value_t gpiohs_get_pin(uint8_t pin)**

Parameter:

Parameter name	Description	Input/Output
pin	Input	Input

Return value: Get the GPIOHS pin value.

4) gpiohs_set_pin_edge

Function: Set the GPIOHS interrupt trigger mode.

Function prototype: **void gpiohs_set_pin_edge(uint8_t pin, gpio_pin_edge_t edge)**

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input
edge	Interrupt trigger mode	Input

Return value: No

5) gpiohs_set_irq

Function: Set the interrupt callback function of GPIOHS.

Function prototype: **void gpiohs_set_irq(uint8_t pin, uint32_t priority, void(*func)()**

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input
priority	Interrupt priority	Input
func	Interrupt callback function	Input

Return value: No

6) gpiohs_irq_register

Function: Set the interrupt callback function of GPIOHS.

Function prototype: **void gpiohs_irq_register(uint8_t pin, uint32_t priority, plic_irq_callback_t callback, void *ctx)**

Parameter:

Parameter name	Description	Input/Output
pin	Input	Input
priority	Interrupt priority	Input
plic_irq_callback_t	Interrupt callback function	Input
ctx	Interrupt callback function	Input

Return value: No

7) gpiohs_irq_unregister

Function: Log off the GPIOHS interrupt.

Function prototype: **void gpiohs_irq_unregister(uint8_t pin)**

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input

Return value: No

Type of data

The related data types and data structures are defined as follows:

- gpio_drive_mode_t: GPIO drive mode
- gpio_pin_value_t: GPIO value
- gpio_pin_edge_t: GPIO edge trigger mode.

1) gpio_drive_mode_t

Function: GPIO drive mode

Define

```
typedef enum _gpio_drive_mode
{
    GPIO_DM_INPUT,
    GPIO_DM_INPUT_PULL_DOWN,
    GPIO_DM_INPUT_PULL_UP,
    GPIO_DM_OUTPUT,
} gpio_drive_mode_t;
```

Member

Member name	Description
GPIO_DM_INPUT	Input
GPIO_DM_INPUT_PULL_DOWN	Input pull_down
GPIO_DM_INPUT_PULL_UP	Input pull_up
GPIO_DM_OUTPUT	Output

2) gpio_pin_value_t

Description: GPIO value.

Define

```
typedef enum _gpio_pin_value
{
        GPIO_PV_LOW,
        GPIO_PV_HIGH
} gpio_pin_value_t;
```

Member

Member name	Description
GPIO_PV_LOW	Low
GPIO_PV_HIGH	High

3) gpio_pin_edge_t

Function: GPIOHS edge trigger mode.

Define

```
typedef enum _gpio_pin_edge
{
        GPIO_PE_NONE,
        GPIO_PE_FALLING,
        GPIO_PE_RISING,
        GPIO_PE_BOTH,
        GPIO_PE_LOW,
        GPIO_PE_HIGH = 8,
} gpio_pin_edge_t;
```

Member

Member name	Description
GPIO_PE_NONE	Not trigger
GPIO_PE_FALLING	negative-edge-triggered
GPIO_PE_RISING	negative-edge-triggered
GPIO_PE_BOTH	double-edge-triggered
GPIO_PE_LOW	low-level-triggered
GPIO_PE_HIGH	high-level-triggered