3.5 Independent button interrupt

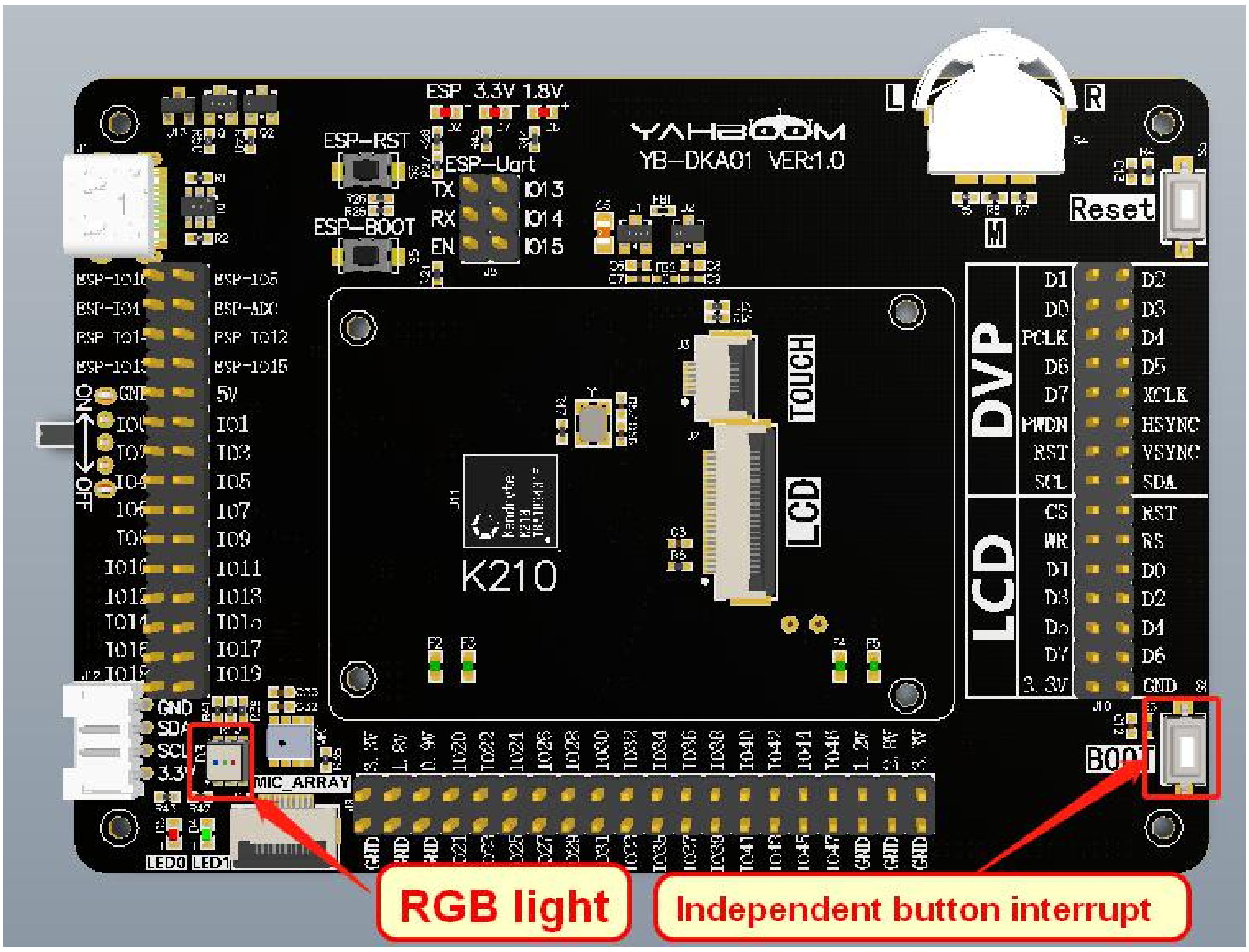
1. Experiment purpose

In this lesson, we mainly learns independent buttons and interrupt functions of K210.

2.Experiment preparation

2.1 components

Independent button BOOT, RGB light.



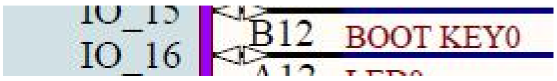
2.2 Component characteristics

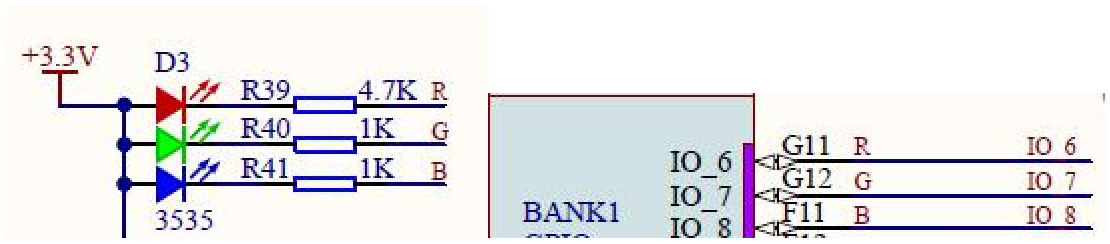
When the independent button BOOT is pressed, IO port is at low level.

When the independent button BOOT is released, IO port is at high level.

3.2 Hardware connection

By default, the K210 development board has already welded the BOOT buttons and RGB lights. The pin connected to the button is IO16. RGB light R is connected to IO6, G is connected to IO7, and B is connected to IO8.





2.4 SDK API function

The header file is plic.h

PLIC can assign any external interrupt source to the external interrupt of each CPU individually. This provides strong flexibility to adapt to different application requirements.

The PLIC module can set an interrupt callback function. When an interrupt is triggered, it will automatically run the interrupt callback function, and the interrupt priority can be configured.

We will provide following interfaces for users:

- plic_init：PLIC Initializes external interrupts.
- plic_irq_enable Enable external interruption.
- plic_irq_disable Disable external interrupts.
- plic_set_priority Set interrupt priority.
- plic_get_priority Gets interrupt priority.
- plic_irq_register Register external interrupt functions.
- plic_irq_deregister Log off the external interrupt function.

3. Experimental principle

When the BOOT button is pressed, this pin with low level. When it is released, this pin with high level.

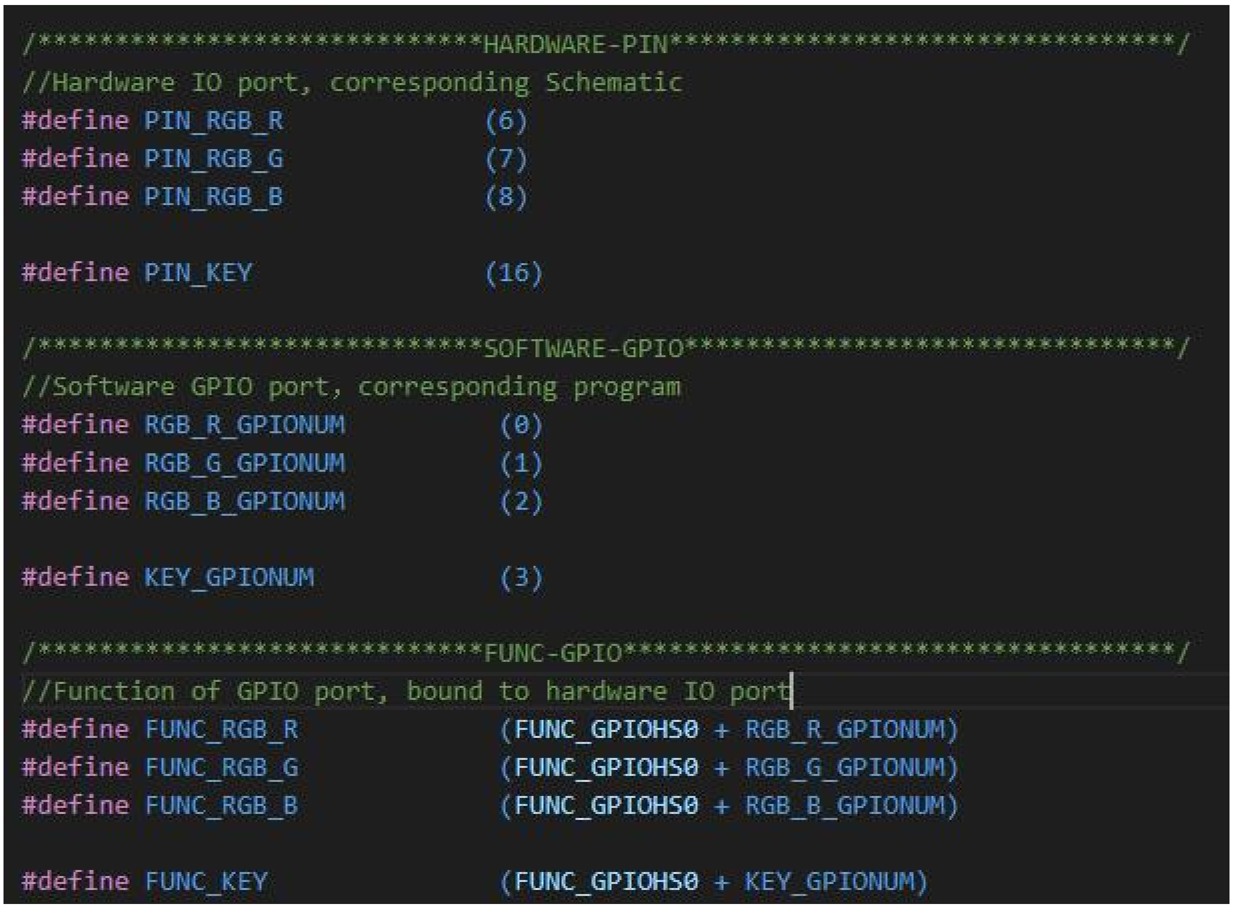
We only need to detect the level of the IO port of the BOOT button. If it is pressed, a falling edge will be generated. If it is released, a rising edge will be generated.

In this way, we can detects and triggers the interrupt of the system, and controls the RGB light to turn on or off.

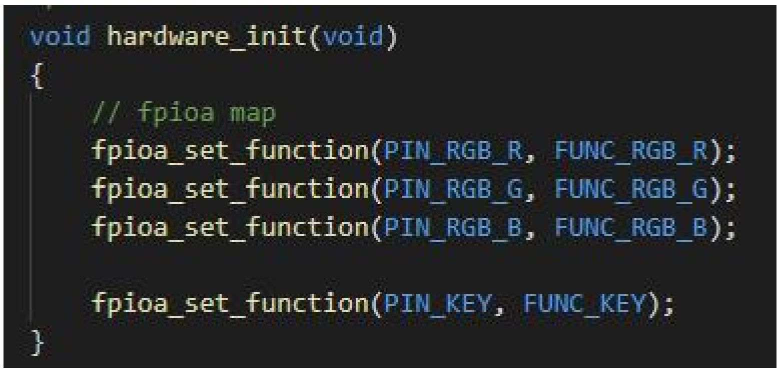
4. Experiment procedure

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.

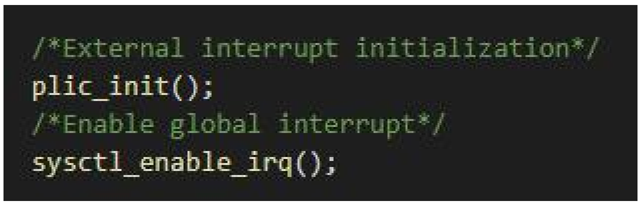


```c
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_RGB_R             (6)
#define PIN_RGB_G             (7)
#define PIN_RGB_B             (8)

#define PIN_KEY               (16)

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

#define KEY_GPIONUM            (3)

/*****************************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)

#define FUNC_KEY               (FUNC_GPIOHS0 + KEY_GPIONUM)
```



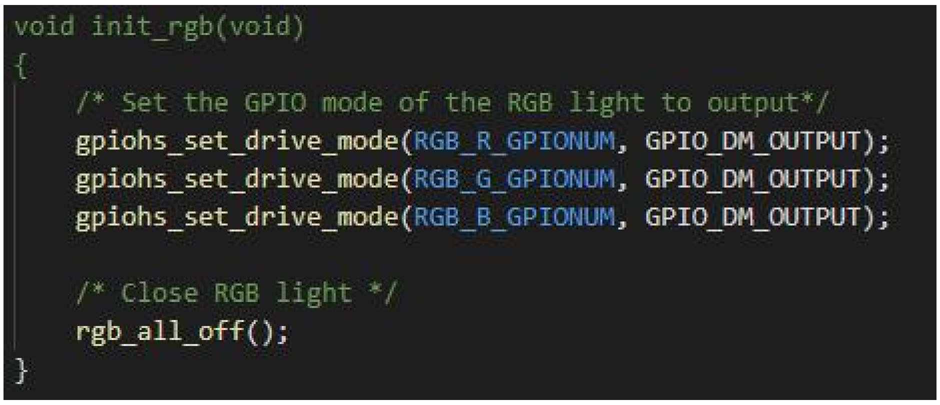
```c
void hardware_init(void)
{
    // fpioa map
    fpioa_set_function(PIN_RGB_R, FUNC_RGB_R);
    fpioa_set_function(PIN_RGB_G, FUNC_RGB_G);
    fpioa_set_function(PIN_RGB_B, FUNC_RGB_B);

    fpioa_set_function(PIN_KEY, FUNC_KEY);
}
```

4.2 Initialize external interrupt service and enable global interrupt. Without this step, the system interrupt will not run, so the interrupt callback function will not be called.



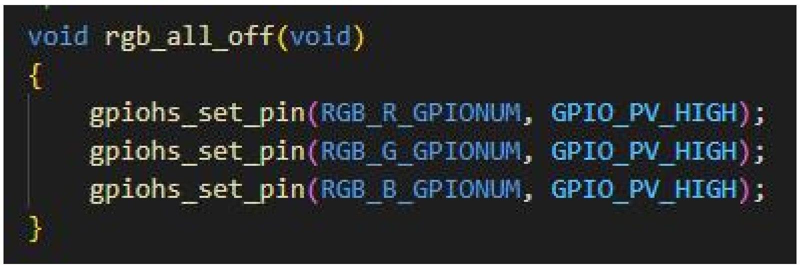
```c
/*External interrupt initialization*/
plic_init();
/*Enable global interrupt*/
sysctl_enable_irq();
```

4.3 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.



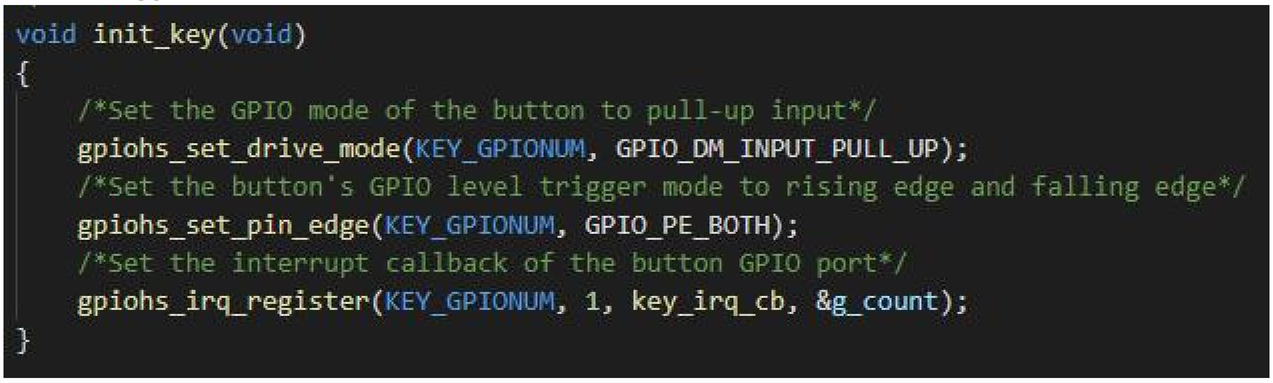
```
void init_rgb(void)
{
    /* Set the GPIO mode of the RGB light to output*/
    gpiohs_set_drive_mode(RGB_R_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_G_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_B_GPIONUM, GPIO_DM_OUTPUT);

    /* Close RGB light */
    rgb_all_off();
}
```

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

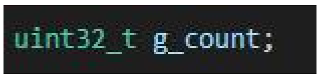


```
void rgb_all_off(void)
{
    gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
}
```

4.5 The BOOT button also needs to be initialized, set the BOOT button to pull-up input mode. Set the GPIO level trigger mode of button to rising edge and falling edge. You can also set single rising edge or single falling edge, etc.,. Setting the BOOT button interrupt callback function Is key_irq_cb. Parameter is g_count, g_count is a global uint32_t variable, and its function is to record the number of level triggers.

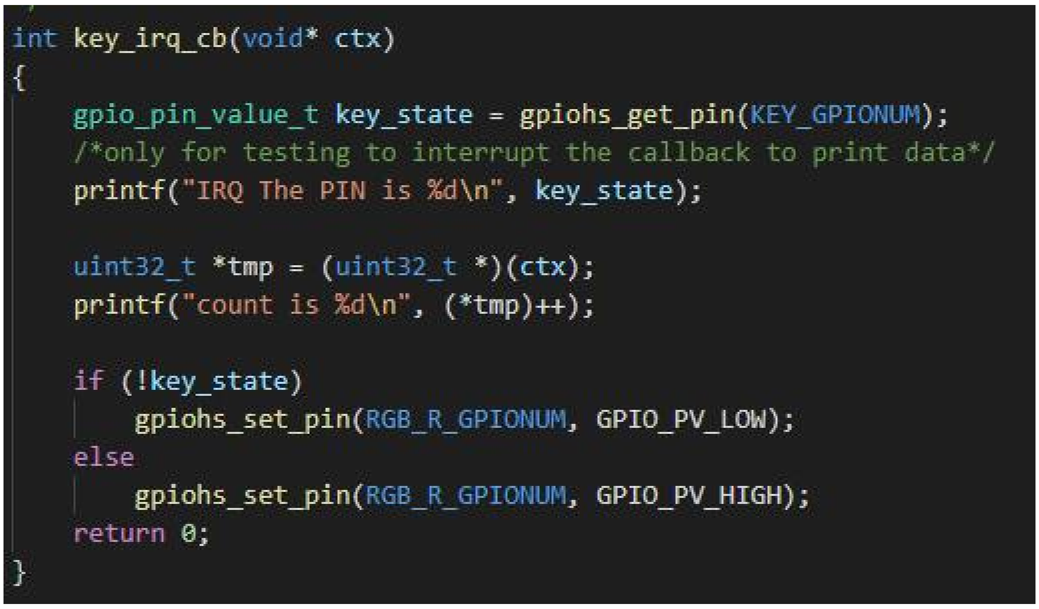


```
void init_key(void)
{
    /*Set the GPIO mode of the button to pull-up input*/
    gpiohs_set_drive_mode(KEY_GPIONUM, GPIO_DM_INPUT_PULL_UP);
    /*Set the button's GPIO level trigger mode to rising edge and falling edge*/
    gpiohs_set_pin_edge(KEY_GPIONUM, GPIO_PE_BOTH);
    /*Set the interrupt callback of the button GPIO port*/
    gpiohs_irq_register(KEY_GPIONUM, 1, key_irq_cb, &g_count);
}
```



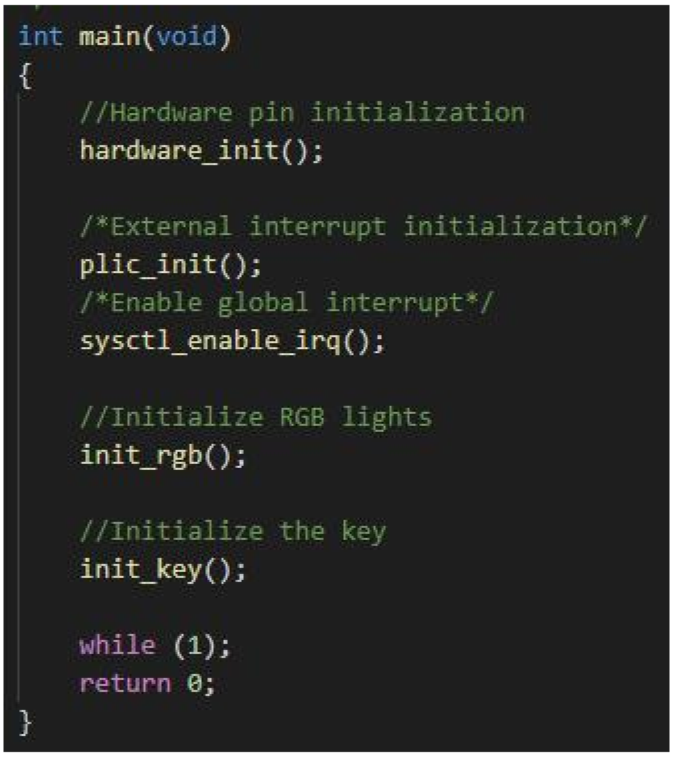
```
uint32_t g_count;
```

4.6 Every time BOOT is pressed or released, the interrupt function key_irq_cb will be triggered. In the interrupt, we need to read the state of the current button is first, saved to key_state, and the current button state is printed out through the serial port. The information is printed in the interrupt only for debugging. It is not recommended in development.
Since ctx is a void pointer, the void pointer means that the pointer type is not specified, so we need to create a new uint32_t type pointer tmp to point to it. Then, the value pointed to by tmp is automatically increased by 1 each time the interrupt is entered, that is, the value of g_count is increased by 1 . Finally, judge the state of the button. When the button is pressed, the red light is on, and when the button is released, the red light is turned off.



```
int key_irq_cb(void* ctx)
{
    gpio_pin_value_t key_state = gpiohs_get_pin(KEY_GPIONUM);
    /*only for testing to interrupt the callback to print data*/
    printf("IRQ The PIN is %d\n", key_state);

    uint32_t *tmp = (uint32_t *)(ctx);
    printf("count is %d\n", (*tmp)++);

    if (!key_state)
        gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_LOW);
    else
        gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
    return 0;
}
```

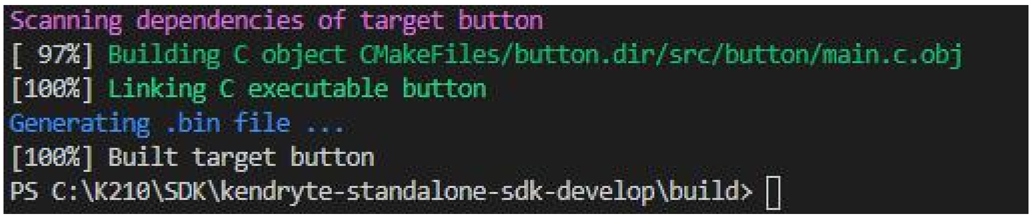
4.7 while loop.



```
int main(void)
{
    //Hardware pin initialization
    hardware_init();

    /*External interrupt initialization*/
    plic_init();
    /*Enable global interrupt*/
    sysctl_enable_irq();

    //Initialize RGB lights
    init_rgb();

    //Initialize the key
    init_key();

    while (1);
    return 0;
}
```

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

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

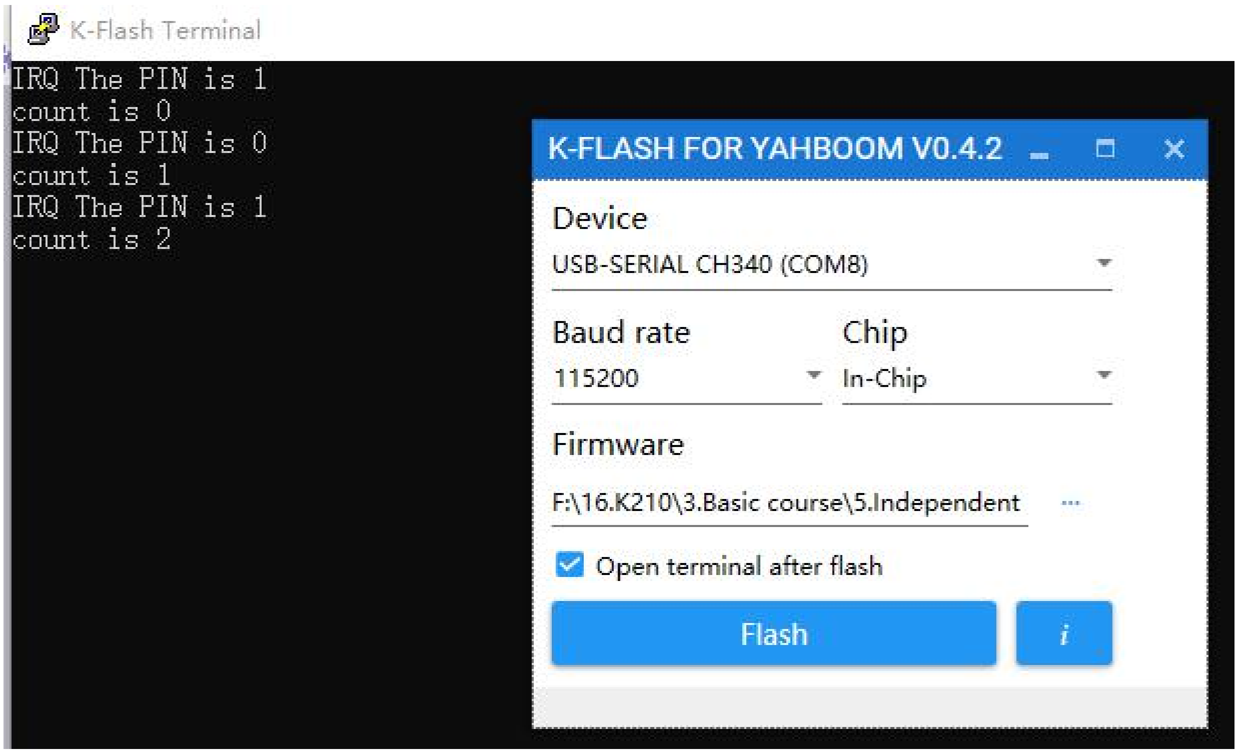


```
Scanning dependencies of target button
[ 97%] Building C object CMakeFiles/button.dir/src/button/main.c.obj
[100%] Linking C executable button
Generating .bin file ...
[100%] Built target button
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

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 button.bin file to the K210 development board.

5. Experimental phenomenon

After the firmware is burned, a terminal interface will pop up. If the terminal interface does not pop up, we can open the serial port assistant to display the debugging content.



K-Flash Terminal

```
IRQ The PIN is 1
count is 0
IRQ The PIN is 0
count is 1
IRQ The PIN is 1
count is 2
```

K-FLASH FOR YAHBOOM V0.4.2

Device: USB-SERIAL CH340 (COM8)

Baud rate: 115200

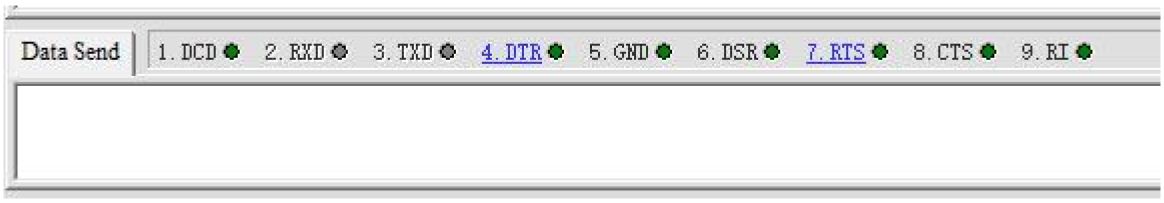
Chip: In-Chip

Firmware: F:\16.K210\3.Basic course\5.Independent

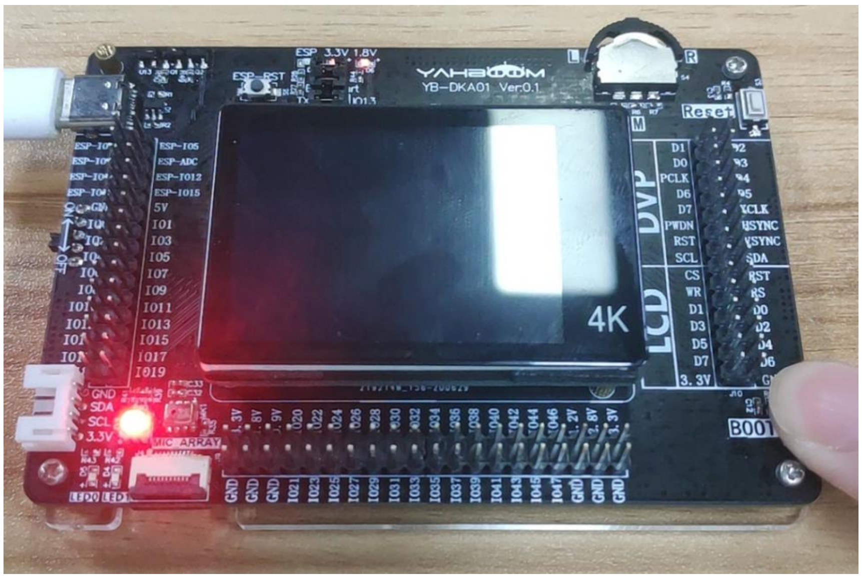
[✓] Open terminal after flash

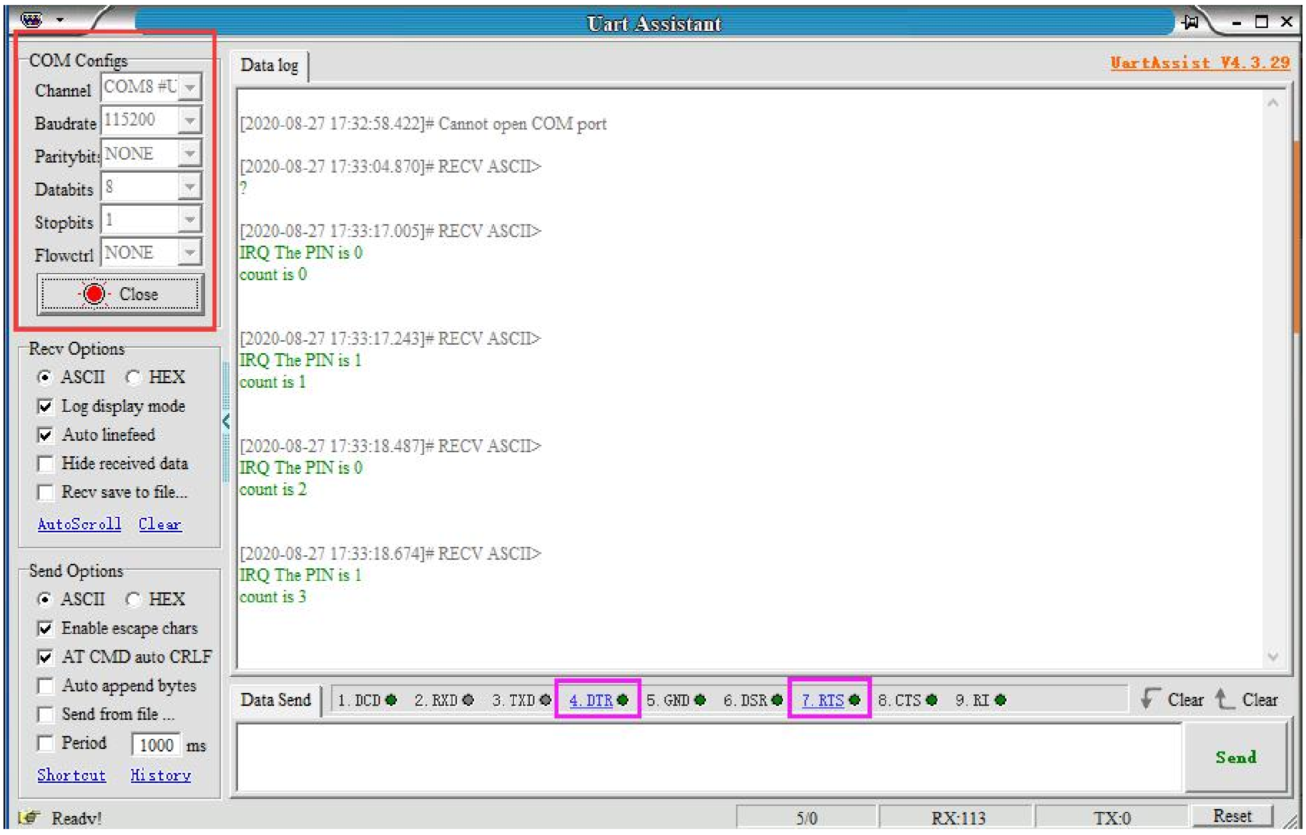
Open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200, and then click to open the serial port assistant.

! Note:
We also need to set the DTR and RTS of the serial port assistant.
At the bottom of the serial port assistant, we can see that 4.DTR and 7.RTS are red by default. Click 4.DTR and 7.RTS to set both to green, and then press the reset button of the K210 development board.



If we press the BOOT button without releasing it, you can see the RGB light is red, and IRQ The PIN is 0, count is 0, they will be displayed on the serial port assistant.
When you release it, it will display IRQ The PIN is 1, count is 1.
Each time you press or when you release it, the value of count is automatically increased by 1.





```
[2020-08-27 17:32:58.422]# Cannot open COM port

[2020-08-27 17:33:04.870]# RECV ASCII>
?

[2020-08-27 17:33:17.005]# RECV ASCII>
IRQ The PIN is 0
count is 0

[2020-08-27 17:33:17.243]# RECV ASCII>
IRQ The PIN is 1
count is 1

[2020-08-27 17:33:18.487]# RECV ASCII>
IRQ The PIN is 0
count is 2

[2020-08-27 17:33:18.674]# RECV ASCII>
IRQ The PIN is 1
count is 3
```

6. Experiment summary

6.1 BOOT button and RGB are also applicable to the function of GPIOHS, but the button uses the input mode and RGB uses the output mode.

2. Before using the external interrupt, we need to initialize the PLIC and enable the global interrupt service.

3. You can pass in a parameter in the interrupt callback function, and the parameter type can pass in the type you need.

Appendix -- API

Header file is plic.h

plic_init

Description: PLIC initializes external interrupts.

Function prototype: **void plic_init(void)**

Parameter: No

Return value: No

plic_irq_enable

Description: Enable external interruption.

Function prototype: **int plic_irq_enable(plic_irq_t irq_number)**

Parameter:

Parameter name	Description	Input/Output
irq_number	IRQ	Input

Return value:

Return value	Description
0	succeed
!0	fail

plic_irq_disable

Description: Disable external interrupts.

Function prototype: **int plic_irq_disable(plic_irq_t irq_number)**

Parameter:

Parameter name	Description	Input/Output
irq_number	IRQ	Input

Return value

Return value	Description
0	succeed
!0	fail

plic_set_priority

Description: Set interrupt priority.

Function prototype: **int plic_set_priority(plic_irq_t irq_number, uint32_t priority)**

Parameter:

Parameter name	Description	Input/Output
irq_number	IRQ	Input
priority	Interrupt Priority	Input

Return value

Return value	Description
0	succeed
!0	fail

plic_get_priority

Description: Gets interrupt priority.

Function prototype: **uint32_t plic_get_priority(plic_irq_t irq_number)**

Parameter:

Parameter name	Description	Input/Output
irq_number	IRQ	Input

Return value: The priority of an IRQ number interrupt

plic_irq_register

Description: Register external interrupt functions.

Function prototype: **int plic_irq_register(plic_irq_t irq, plic_irq_callback_t callback, void* ctx)**

Parameter:

Parameter name	Description	Input/Output
irq	IRQ	Input
callback	Interrupt callback function	Input
ctx	The argument to the callback function	Input

Return value:

Return value	Description
0	succeed
!0	fail

plic_irq_deregister

Description: Log off the external interrupt function.

Function prototype: int plic_irq_deregister(plic_irq_t irq)

Parameter:

Parameter name	Description	Input/Output
irq	IRQ	Input

Return value:

Return value	Description
0	succeed
!0	fail

Eg:

```
/*Set the triggering interrupt for GPIOHS0*/
int count = 0;
int gpiohs_pin_onchange_isr(void *ctx)
{
        int *userdata = (int *)ctx;
        *userdata++;
}
plic_init();
plic_set_priority(IRQN_GPIOHS0_INTERRUPT, 1);
plic_irq_register(IRQN_GPIOHS0_INTERRUPT, gpiohs_pin_onchange_isr, &count);
plic_irq_enable(IRQN_GPIOHS0_INTERRUPT);
sysctl_enable_irq();
```

Data type

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

plic_irq_t: external interrupt number.

plic_irq_callback_t: external interrupt callback function.

plic_irq_t

Description: External interrupt number.

Definition

```
typedef enum _plic_irq
{
        IRQN_NO_INTERRUPT = 0, /*!< The non-existent interrupt */
        IRQN_SPI0_INTERRUPT          = 1, /*!< SPI0 interrupt */
        IRQN_SPI1_INTERRUPT          = 2, /*!< SPI1 interrupt */
        IRQN_SPI_SLAVE_INTERRUPT = 3, /*!< SPI_SLAVE interrupt */
        IRQN_SPI3_INTERRUPT          = 4, /*!< SPI3 interrupt */
        IRQN_I2S0_INTERRUPT          = 5, /*!< I2S0 interrupt */
        IRQN_I2S1_INTERRUPT          = 6, /*!< I2S1 interrupt */
        IRQN_I2S2_INTERRUPT          = 7, /*!< I2S2 interrupt */
        IRQN_I2C0_INTERRUPT          = 8, /*!< I2C0 interrupt */
        IRQN_I2C1_INTERRUPT          = 9, /*!< I2C1 interrupt */
        IRQN_I2C2_INTERRUPT          = 10, /*!< I2C2 interrupt */
        IRQN_UART1_INTERRUPT          = 11, /*!< UART1 interrupt */
        IRQN_UART2_INTERRUPT          = 12, /*!< UART2 interrupt */
        IRQN_UART3_INTERRUPT          = 13, /*!< UART3 interrupt */
        IRQN_TIMER0A_INTERRUPT      = 14, /*!< TIMER0 channel 0 or 1 interrupt */
        IRQN_TIMER0B_INTERRUPT      = 15, /*!< TIMER0 channel 2 or 3 interrupt */
        IRQN_TIMER1A_INTERRUPT      = 16, /*!< TIMER1 channel 0 or 1 interrupt */
        IRQN_TIMER1B_INTERRUPT      = 17, /*!< TIMER1 channel 2 or 3 interrupt */
        IRQN_TIMER2A_INTERRUPT      = 18, /*!< TIMER2 channel 0 or 1 interrupt */
        IRQN_TIMER2B_INTERRUPT      = 19, /*!< TIMER2 channel 2 or 3 interrupt */
        IRQN_RTC_INTERRUPT            = 20, /*!< RTC tick and alarm interrupt */
        IRQN_WDT0_INTERRUPT            = 21, /*!< Watching dog timer0 interrupt */
        IRQN_WDT1_INTERRUPT            = 22, /*!< Watching dog timer1 interrupt */
        IRQN_APB_GPIO_INTERRUPT    = 23, /*!< APB GPIO interrupt */
        IRQN_DVP_INTERRUPT            = 24, /*!< Digital video port interrupt */
        IRQN_AI_INTERRUPT            = 25, /*!< AI accelerator interrupt */
        IRQN_FFT_INTERRUPT           = 26, /*!< FFT accelerator interrupt */
        IRQN_DMA0_INTERRUPT            = 27, /*!< DMA channel0 interrupt */
        IRQN_DMA1_INTERRUPT            = 28, /*!< DMA channel1 interrupt */
        IRQN_DMA2_INTERRUPT            = 29, /*!< DMA channel2 interrupt */
        IRQN_DMA3_INTERRUPT            = 30, /*!< DMA channel3 interrupt */
        IRQN_DMA4_INTERRUPT            = 31, /*!< DMA channel4 interrupt */
        IRQN_DMA5_INTERRUPT            = 32, /*!< DMA channel5 interrupt */
        IRQN_UARTHS_INTERRUPT         = 33, /*!< Hi-speed UART0 interrupt */
        IRQN_GPIOHS0_INTERRUPT      = 34, /*!< Hi-speed GPIO0 interrupt */
        IRQN_GPIOHS1_INTERRUPT      = 35, /*!< Hi-speed GPIO1 interrupt */
        IRQN_GPIOHS2_INTERRUPT      = 36, /*!< Hi-speed GPIO2 interrupt */
        IRQN_GPIOHS3_INTERRUPT      = 37, /*!< Hi-speed GPIO3 interrupt */
        IRQN_GPIOHS4_INTERRUPT      = 38, /*!< Hi-speed GPIO4 interrupt */
        IRQN_GPIOHS5_INTERRUPT      = 39, /*!< Hi-speed GPIO5 interrupt */
        IRQN_GPIOHS6_INTERRUPT      = 40, /*!< Hi-speed GPIO6 interrupt */
```

```
    IRQN_GPIOHS7_INTERRUPT     = 41, /*!< Hi-speed GPIO7 interrupt */
    IRQN_GPIOHS8_INTERRUPT     = 42, /*!< Hi-speed GPIO8 interrupt */
    IRQN_GPIOHS9_INTERRUPT     = 43, /*!< Hi-speed GPIO9 interrupt */
    IRQN_GPIOHS10_INTERRUPT    = 44, /*!< Hi-speed GPIO10 interrupt */
    IRQN_GPIOHS11_INTERRUPT    = 45, /*!< Hi-speed GPIO11 interrupt */
    IRQN_GPIOHS12_INTERRUPT    = 46, /*!< Hi-speed GPIO12 interrupt */
    IRQN_GPIOHS13_INTERRUPT    = 47, /*!< Hi-speed GPIO13 interrupt */
    IRQN_GPIOHS14_INTERRUPT    = 48, /*!< Hi-speed GPIO14 interrupt */
    IRQN_GPIOHS15_INTERRUPT    = 49, /*!< Hi-speed GPIO15 interrupt */
    IRQN_GPIOHS16_INTERRUPT    = 50, /*!< Hi-speed GPIO16 interrupt */
    IRQN_GPIOHS17_INTERRUPT    = 51, /*!< Hi-speed GPIO17 interrupt */
    IRQN_GPIOHS18_INTERRUPT    = 52, /*!< Hi-speed GPIO18 interrupt */
    IRQN_GPIOHS19_INTERRUPT    = 53, /*!< Hi-speed GPIO19 interrupt */
    IRQN_GPIOHS20_INTERRUPT    = 54, /*!< Hi-speed GPIO20 interrupt */
    IRQN_GPIOHS21_INTERRUPT    = 55, /*!< Hi-speed GPIO21 interrupt */
    IRQN_GPIOHS22_INTERRUPT    = 56, /*!< Hi-speed GPIO22 interrupt */
    IRQN_GPIOHS23_INTERRUPT    = 57, /*!< Hi-speed GPIO23 interrupt */
    IRQN_GPIOHS24_INTERRUPT    = 58, /*!< Hi-speed GPIO24 interrupt */
    IRQN_GPIOHS25_INTERRUPT    = 59, /*!< Hi-speed GPIO25 interrupt */
    IRQN_GPIOHS26_INTERRUPT    = 60, /*!< Hi-speed GPIO26 interrupt */
    IRQN_GPIOHS27_INTERRUPT    = 61, /*!< Hi-speed GPIO27 interrupt */
    IRQN_GPIOHS28_INTERRUPT    = 62, /*!< Hi-speed GPIO28 interrupt */
    IRQN_GPIOHS29_INTERRUPT    = 63, /*!< Hi-speed GPIO29 interrupt */
    IRQN_GPIOHS30_INTERRUPT    = 64, /*!< Hi-speed GPIO30 interrupt */
    IRQN_GPIOHS31_INTERRUPT    = 65, /*!< Hi-speed GPIO31 interrupt */
    IRQN_MAX
} plic_irq_t;
```

Member

Member name	**Description**
IRQN_NO_INTERRUPT	not exist
IRQN_SPI0_INTERRUPT	SPI0 interrupt
IRQN_SPI1_INTERRUPT	SPI1 interrupt
IRQN_SPI_SLAVE_INTERRUPT	From the SPI interrupt
IRQN_SPI3_INTERRUPT	From the SPI3 interrupt
IRQN_I2S0_INTERRUPT	I2S0 interrupt
IRQN_I2S1_INTERRUPT	I2S1 interrupt
IRQN_I2S2_INTERRUPT	I2S2 interrupt
IRQN_I2C0_INTERRUPT	I2C0 interrupt
IRQN_I2C1_INTERRUPT	I2C1 interrupt
IRQN_I2C2_INTERRUPT	I2C2 interrupt
IRQN_UART1_INTERRUPT	UART1 interrupt

Member name	Description
IRQN_UART2_INTERRUPT	UART2 interrupt
IRQN_UART3_INTERRUPT	UART3 interrupt
IRQN_TIMER0A_INTERRUPT	TIMER0 channel 0 and 1 interrupt
IRQN_TIMER0B_INTERRUPT	TIMER0 channel 2 and 3 interrupt
IRQN_TIMER1A_INTERRUPT	TIMER1 channel 0 and 1 interrupt
IRQN_TIMER1B_INTERRUPT	TIMER1 channel 2 and 3 interrupt
IRQN_TIMER2A_INTERRUPT	TIMER2 channel 0 and 1 interrupt
IRQN_TIMER2B_INTERRUPT	TIMER2 channel 2 and 3 interrupt
IRQN_RTC_INTERRUPT	RTC tick interrupt and alarm interrupt
IRQN_WDT0_INTERRUPT	Watchdog 0 interrupt
IRQN_WDT1_INTERRUPT	Watchdog 1 interrupt
IRQN_APB_GPIO_INTERRUPT	General GPIO interrupt
IRQN_DVP_INTERRUPT	DVP interrupt
IRQN_AI_INTERRUPT	AI Accelerator interrupt
IRQN_FFT_INTERRUPTFFT	Fourier accelerator interrupt
IRQN_DMA0_INTERRUPT	DMA channel 0 interrupt
IRQN_DMA1_INTERRUPT	DMA channel 1 interrupt
IRQN_DMA2_INTERRUPT	DMA channel 2 interrupt
IRQN_DMA3_INTERRUPT	DMA channel 3 interrupt
IRQN_DMA4_INTERRUPT	DMA channel 4 interrupt
IRQN_DMA5_INTERRUPT	DMA channel 5 interrupt
IRQN_UARTHS_INTERRUPT	High-speed UART interrupt
IRQN_GPIOHS0_INTERRUPT	High-speed GPIO0 interrupt
IRQN_GPIOHS1_INTERRUPT	High-speed GPIO1 interrupt
IRQN_GPIOHS2_INTERRUPT	High-speed GPIO2 interrupt
IRQN_GPIOHS3_INTERRUPT	High-speed GPIO3 interrupt
IRQN_GPIOHS4_INTERRUPT	High-speed GPIO4 interrupt
IRQN_GPIOHS5_INTERRUPT	High-speed GPIO5 interrupt
IRQN_GPIOHS6_INTERRUPT	High-speed GPIO6 interrupt
IRQN_GPIOHS7_INTERRUPT	High-speed GPIO7 interrupt
IRQN_GPIOHS8_INTERRUPT	High-speed GPIO8 interrupt
IRQN_GPIOHS9_INTERRUPT	High-speed GPIO9 interrupt
IRQN_GPIOHS10_INTERRUPT	High-speed GPIO10 interrupt
IRQN_GPIOHS11_INTERRUPT	High-speed GPIO11 interrupt
IRQN_GPIOHS12_INTERRUPT	High-speed GPIO12 interrupt
IRQN_GPIOHS13_INTERRUPT	High-speed GPIO13 interrupt
IRQN_GPIOHS14_INTERRUPT	High-speed GPIO14 interrupt

Member name	**Description**
IRQN_GPIOHS15_INTERRUPT	High-speed GPIO15 interrupt
IRQN_GPIOHS16_INTERRUPT	High-speed GPIO16 interrupt
IRQN_GPIOHS17_INTERRUPT	High-speed GPIO17 interrupt
IRQN_GPIOHS18_INTERRUPT	High-speed GPIO18 interrupt
IRQN_GPIOHS19_INTERRUPT	High-speed GPIO19 interrupt
IRQN_GPIOHS20_INTERRUPT	High-speed GPIO20 interrupt
IRQN_GPIOHS21_INTERRUPT	High-speed GPIO21 interrupt
IRQN_GPIOHS22_INTERRUPT	High-speed GPIO22 interrupt
IRQN_GPIOHS23_INTERRUPT	High-speed GPIO23 interrupt
IRQN_GPIOHS24_INTERRUPT	High-speed GPIO24 interrupt
IRQN_GPIOHS25_INTERRUPT	High-speed GPIO25 interrupt
IRQN_GPIOHS26_INTERRUPT	High-speed GPIO26 interrupt
IRQN_GPIOHS27_INTERRUPT	High-speed GPIO27 interrupt
IRQN_GPIOHS28_INTERRUPT	High-speed GPIO28 interrupt
IRQN_GPIOHS29_INTERRUPT	High-speed GPIO29 interrupt
IRQN_GPIOHS30_INTERRUPT	High-speed GPIO30 interrupt
IRQN_GPIOHS31_INTERRUPT	High-speed GPIO31 interrupt

plic_irq_callback_t

Description: External interrupt callback function.

Define

typedef int (*plic_irq_callback_t)(void *ctx);