# 3.5 独立按键中断实验

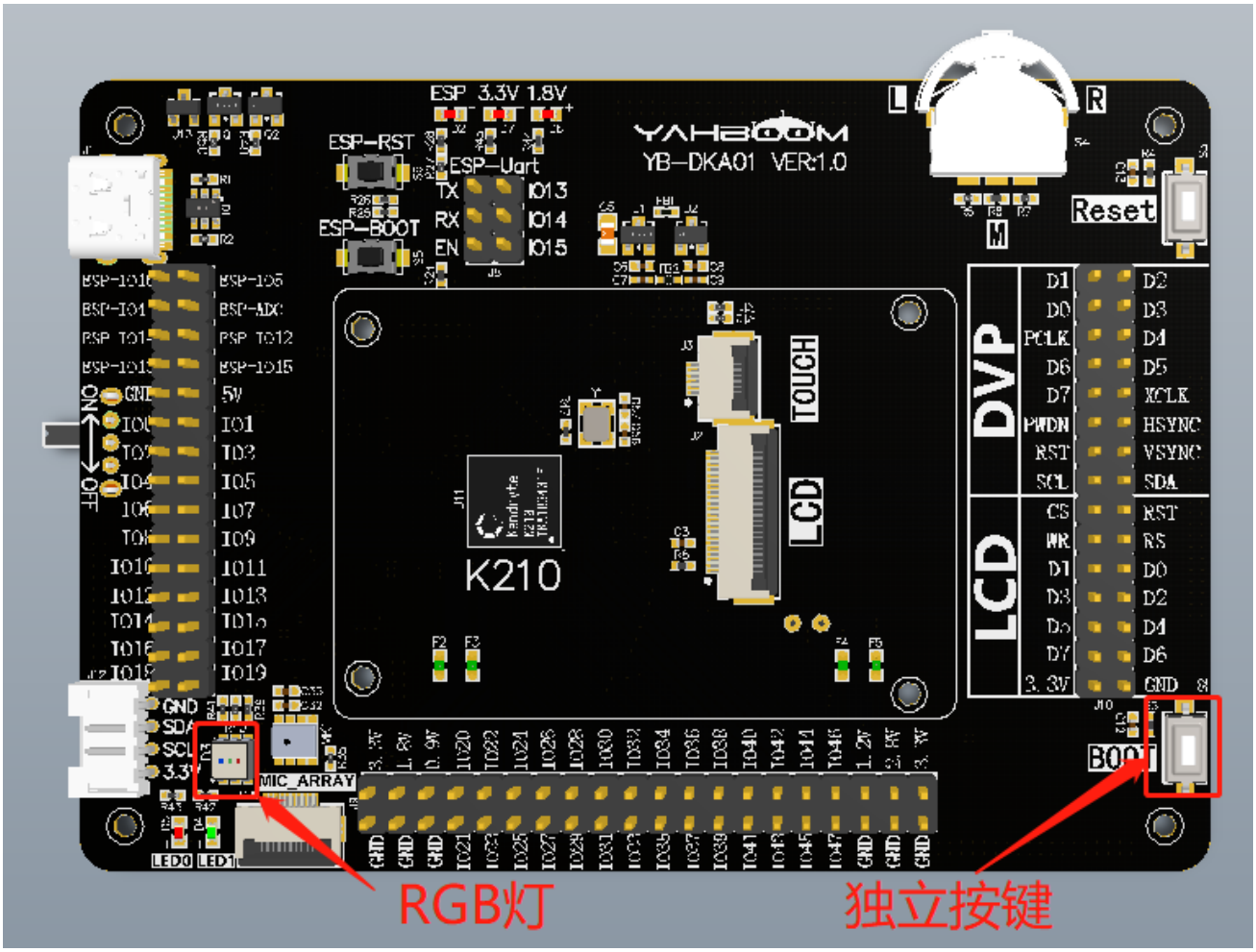
## 一、实验目的

本节课主要学习 K210 的独立按键以及中断的功能。

## 二、实验准备

### 1.实验元件

独立按键 BOOT、RGB 灯

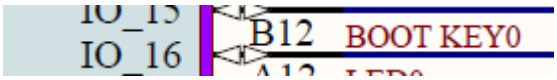


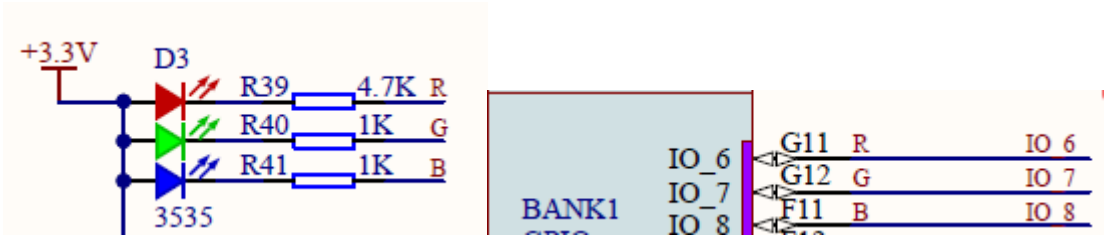

### 2.元件特性

独立按键 BOOT 按下时 IO 口为低电平，松开为高电平。

### 3.硬件连接

K210开发板出厂默认已经焊接好BOOT按键和RGB灯。按键连接的引脚为IO16。RGB灯R连接的是IO6，G连接的是IO7，B连接的是IO8。




4.SDK中对应API功能

对应的头文件 plic.h

PLIC 可以将任一外部中断源单独分配到每个 CPU 的外部中断上。这提供了强大的灵活性，能适应不同的应用需求。

PLIC模块可以设置中断回调函数，当触发中断时，会自动运行中断回调函数，并且可以配置中断优先级。

为用户提供以下接口：

- plic_init：PLIC初始化外部中断。
- plic_irq_enable：使能外部中断。
- plic_irq_disable：禁用外部中断。
- plic_set_priority：设置中断优先级。
- plic_get_priority：获取中断优先级。
- plic_irq_register：注册外部中断函数。
- plic_irq_deregister：注销外部中断函数。

三、实验原理

BOOT按键按下的时候会把电平拉低，松开的时候会把电平拉高，只需要检测BOOT按键的IO口的电平，如果是按下则会产生下降沿，松开会产生上升沿，以此的方式来检测并触发系统的中断，然后控制RGB灯亮或者灭。

四、实验过程

1.首先根据上面的硬件连接引脚图，K210 的硬件引脚和软件功能使用的是 FPIOA 映射关系。

这里要注意的是程序里操作的都是软件引脚，所以需要先把硬件引脚映射成软件 GPIO 功能，操作的时候直接操作软件 GPIO 即可。

```
/*****************************HARDWARE-PIN*************************
// 硬件IO口，与原理图对应
#define PIN_RGB_R               (6)
#define PIN_RGB_G               (7)
#define PIN_RGB_B               (8)

#define PIN_KEY                 (16)

/*****************************SOFTWARE-GPIO************************
// 软件GPIO口，与程序对应
#define RGB_R_GPIONUM           (0)
#define RGB_G_GPIONUM           (1)
#define RGB_B_GPIONUM           (2)

#define KEY_GPIONUM             (3)

/*****************************FUNC-GPIO****************************
// GPIO口的功能，绑定到硬件IO口
#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)
```

```
void hardware_init(void)
{
    // fpioa映射
    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);
}
```

2.第二部需要初始化外部中断服务，并且使能全局中断。如果没有这一步操

作，系统的中断就不会运行，所以也不会调用中断回调函数。

```
/* 外部中断初始化 */
plic_init();
/* 使能全局中断 */
sysctl_enable_irq();
```

3.在使用 RGB 灯前需要初始化，也就是把 RGB 灯的软件 GPIO 设置为输出模式。

```
void init_rgb(void)
{
    // 设置RGB灯的GPIO模式为输出
    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);

    // 关闭RGB灯
    rgb_all_off();
}
```

4.然后关闭 RGB 灯，同样是设置 RGB 灯的 GPIO 为高电平则可以让 RGB 灯熄灭。

```
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);
}
```

5.使用 BOOT 按键同样需要初始化，设置 BOOT 键为上拉输入模式，设置按键的 GPIO 电平触发模式为上升沿和下降沿，也可以设置单上升沿或单下降沿等，设置 BOOT 按键的中断回调函数为 key_irq_cb，参数为 g_count，g_count 是一个全局 uint32_t 变量，功能是记录电平触发的次数。

```
uint32_t g_count;
```

```c
void init_key(void)
{
    /* 设置按键的GPIO模式为上拉输入 */
    gpiohs_set_drive_mode(KEY_GPIONUM, GPIO_DM_INPUT_PULL_UP);
    /* 设置按键的GPIO电平触发模式为上升沿和下降沿 */
    gpiohs_set_pin_edge(KEY_GPIONUM, GPIO_PE_BOTH);
    /* 设置按键GPIO口的中断回调 */
    gpiohs_irq_register(KEY_GPIONUM, 1, key_irq_cb, &g_count);
}
```

6.每次 BOOT 按下或者松开都会触发中断函数 key_irq_cb，在中断里先读取当前按键的状态，保存到 key_state 中，并且通过串口打印出当前按键的状态，这里只是为了调试才在中断中打印信息的，在后续的开发中不建议这样操作。由于 ctx 为 void 型指针，void 型指针表示不指定指针类型，所以我们需要新建一个 uint32_t 类型的指针 tmp 指向它，然后每次进中断 tmp 所指向的值自动加 1，也就是 g_count 的值加 1。最后是判断按键的状态，当按键为按下时，点亮红灯，当按键为松开时，关闭红灯。

```c
int key_irq_cb(void* ctx)
{
    gpio_pin_value_t key_state = gpiohs_get_pin(KEY_GPIONUM);
    /* 这里只是为了测试才在中断回调打印数据，正常情况下是不建议这么做的。*/
    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;
}
```

7.最后是一个 while(1)循环，这个是必须的，否则系统就会退出，不再运行。

```c
int main(void)
{
    // 硬件引脚初始化
    hardware_init();

    plic_init();
    sysctl_enable_irq();

    // 初始化RGB灯
    init_rgb();

    // 初始化按键key
    init_key();

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

8.编译调试，烧录运行

把本课程资料中的 button 复制到 SDK 中的 src 目录下，

然后进入 build 目录，运行以下命令编译。

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>
```

编译完成后，在 build 文件夹下会生成 button.bin 文件。

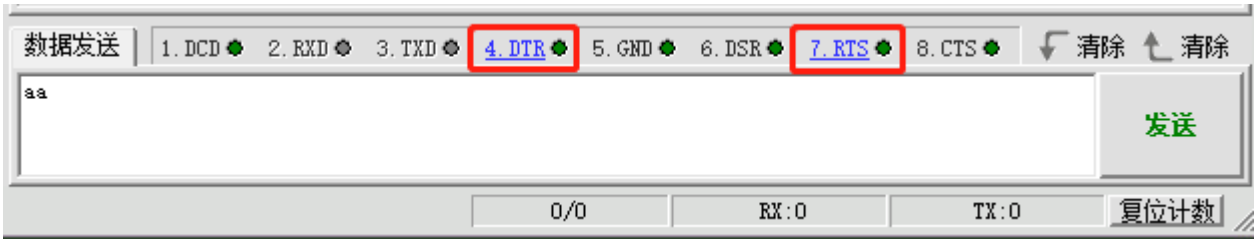
使用 type-C 数据线连接电脑与 K210 开发板，打开 kflash，选择对应的设备，再将程序固件烧录到 K210 开发板上。

## 五、实验现象

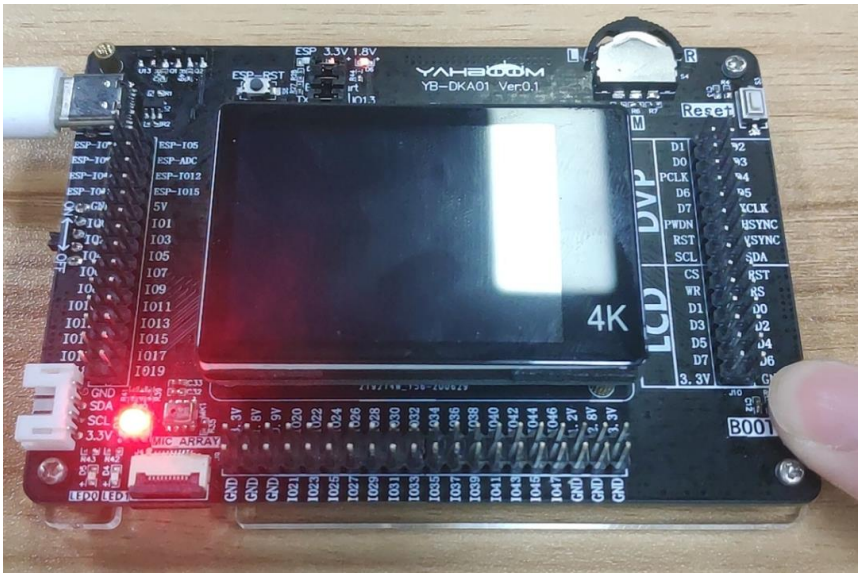
烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以

打开串口助手显示调试内容。

打开电脑的串口助手，选择对应的 K210 开发板对应的串口号，波特率设置为 115200,然后点击打开串口助手。注意还需要设置一下串口助手的 DTR 和 RTS。在串口助手底部此时的 4.DTR 和 7.RTS 默认是红色的，点击 4.DTR 和 7.RTS，都设置为绿色，然后按一下 K210 开发板的复位键。




按下 BOOT 按键不松开，可以看到 RGB 亮红灯，并且串口助手上显示 IRQ The PIN is 0  count is 0，松开的时候会显示 IRQ The PIN is 1  count is 1，每次按下或松开 count 的值都自动加 1。




## 六、实验总结

1.BOOT 按键与 RGB 同样适用 GPIOHS 的函数，只是按键使用输入模式，RGB 使用输出模式。

2.使用外部中断前需要先初始化 PLIC 以及使能全局中断服务。

3.在中断回调函数中可以传入一个参数，参数类型可以传入自己需要的类型。

附：API

对应的头文件 plic.h

## plic_init

### 描述

PLIC 初始化外部中断。

### 函数原型

```
void plic_init(void)
```

### 参数

无。

### 返回值

无。

## plic_irq_enable

### 描述

使能外部中断。

### 函数原型

```
int plic_irq_enable(plic_irq_t irq_number)
```

### 参数

| 参数名称 | 描述 | 输入输出 |
| --- | --- | --- |
| irq_number | 中断号 | 输入 |

### 返回值

| 返回值 | 描述 |
| --- | --- |
| 0 | 成功 |
| 非 0 | 失败 |

## plic_irq_disable

### 描述

禁用外部中断。

### 函数原型

```
int plic_irq_disable(plic_irq_t irq_number)
```

### 参数

| 参数名称 | 描述 | 输入输出 |
| --- | --- | --- |
| irq_number | 中断号 | 输入 |

### 返回值

| 返回值 | 描述 |
| --- | --- |
| 0 | 成功 |
| 非 0 | 失败 |

## plic_set_priority

### 描述

设置中断优先级。

### 函数原型

```
int plic_set_priority(plic_irq_t irq_number, uint32_t priority)
```

### 参数

| 参数名称 | 描述 | 输入输出 |
| --- | --- | --- |
| irq_number | 中断号 | 输入 |
| priority | 中断优先级 | 输入 |

### 返回值

| 返回值 | 描述 |
| --- | --- |
| 0 | 成功 |
| 非 0 | 失败 |

## plic_get_priority

### 描述

获取中断优先级。

### 函数原型

```
uint32_t plic_get_priority(plic_irq_t irq_number)
```

### 参数

**参数名称　描述　输入输出**
irq_number 中断号 输入

### 返回值

irq_number 中断的优先级。

## plic_irq_register

### 描述

注册外部中断函数。

### 函数原型

```
int plic_irq_register(plic_irq_t irq, plic_irq_callback_t callback, void* ctx)
```

### 参数

| 参数名称 | 描述 | 输入输出 |
| --- | --- | --- |
| irq | 中断号 | 输入 |
| callback | 中断回调函数 | 输入 |
| ctx | 回调函数的参数 | 输入 |

### 返回值

| 返回值 | 描述 |
| --- | --- |
| 0 | 成功 |
| 非 0 | 失败 |

## plic_irq_deregister

### 描述

注销外部中断函数。

### 函数原型

```
int plic_irq_deregister(plic_irq_t irq)
```

### 参数

| 参数名称 | 描述 | 输入输出 |
| --- | --- | --- |
| irq | 中断号 | 输入 |

### 返回值

| 返回值 | 描述 |
| --- | --- |
| 0 | 成功 |
| 非 0 | 失败 |

### 举例

```
/* 设置 GPIOHS0 的触发中断 */
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();
```

## 数据类型

相关数据类型、数据结构定义如下：

- plic_irq_t：外部中断号。
- plic_irq_callback_t：外部中断回调函数。

## plic_irq_t

### 描述

外部中断号。

定义

```
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;
```

**成员**

| 成员名称 | 描述 |
| --- | --- |
| IRQN_NO_INTERRUPT | 不存在 |
| IRQN_SPI0_INTERRUPT | SPI0 中断 |
| IRQN_SPI1_INTERRUPT | SPI1 中断 |
| IRQN_SPI_SLAVE_INTERRUPT | 从 SPI 中断 |
| IRQN_SPI3_INTERRUPT | SPI3 中断 |
| IRQN_I2S0_INTERRUPT | I2S0 中断 |
| IRQN_I2S1_INTERRUPT | I2S1 中断 |
| IRQN_I2S2_INTERRUPT | I2S2 中断 |
| IRQN_I2C0_INTERRUPT | I2C0 中断 |
| IRQN_I2C1_INTERRUPT | I2C1 中断 |

| 成员名称 | 描述 |
| --- | --- |
| IRQN_I2C2_INTERRUPT | I2C2 中断 |
| IRQN_UART1_INTERRUPT | UART1 中断 |
| IRQN_UART2_INTERRUPT | UART2 中断 |
| IRQN_UART3_INTERRUPT | UART3 中断 |
| IRQN_TIMER0A_INTERRUPT | TIMER0 通道 0 和 1 中断 |
| IRQN_TIMER0B_INTERRUPT | TIMER0 通道 2 和 3 中断 |
| IRQN_TIMER1A_INTERRUPT | TIMER1 通道 0 和 1 中断 |
| IRQN_TIMER1B_INTERRUPT | TIMER1 通道 2 和 3 中断 |
| IRQN_TIMER2A_INTERRUPT | TIMER2 通道 0 和 1 中断 |
| IRQN_TIMER2B_INTERRUPT | TIMER2 通道 2 和 3 中断 |
| IRQN_RTC_INTERRUPT | RTC 滴答中断和报警中断 |
| IRQN_WDT0_INTERRUPT | 看门狗 0 中断 |
| IRQN_WDT1_INTERRUPT | 看门狗 1 中断 |
| IRQN_APB_GPIO_INTERRUPT | 普通 GPIO 中断 |
| IRQN_DVP_INTERRUPT | 数字摄像头（DVP）中断 |
| IRQN_AI_INTERRUPT | AI 加速器中断 |
| IRQN_FFT_INTERRUPTFFT | 傅里叶加速器中断 |
| IRQN_DMA0_INTERRUPT | DMA 通道 0 中断 |
| IRQN_DMA1_INTERRUPT | DMA 通道 1 中断 |
| IRQN_DMA2_INTERRUPT | DMA 通道 2 中断 |
| IRQN_DMA3_INTERRUPT | DMA 通道 3 中断 |
| IRQN_DMA4_INTERRUPT | DMA 通道 4 中断 |
| IRQN_DMA5_INTERRUPT | DMA 通道 5 中断 |
| IRQN_UARTHS_INTERRUPT | 高速 UART 中断 |
| IRQN_GPIOHS0_INTERRUPT | 高速 GPIO0 中断 |
| IRQN_GPIOHS1_INTERRUPT | 高速 GPIO1 中断 |
| IRQN_GPIOHS2_INTERRUPT | 高速 GPIO2 中断 |
| IRQN_GPIOHS3_INTERRUPT | 高速 GPIO3 中断 |
| IRQN_GPIOHS4_INTERRUPT | 高速 GPIO4 中断 |
| IRQN_GPIOHS5_INTERRUPT | 高速 GPIO5 中断 |
| IRQN_GPIOHS6_INTERRUPT | 高速 GPIO6 中断 |
| IRQN_GPIOHS7_INTERRUPT | 高速 GPIO7 中断 |
| IRQN_GPIOHS8_INTERRUPT | 高速 GPIO8 中断 |
| IRQN_GPIOHS9_INTERRUPT | 高速 GPIO9 中断 |
| IRQN_GPIOHS10_INTERRUPT | 高速 GPIO10 中断 |
| IRQN_GPIOHS11_INTERRUPT | 高速 GPIO11 中断 |
| IRQN_GPIOHS12_INTERRUPT | 高速 GPIO12 中断 |

| 成员名称 | 描述 |
| --- | --- |
| IRQN_GPIOHS13_INTERRUPT | 高速 GPIO13 中断 |
| IRQN_GPIOHS14_INTERRUPT | 高速 GPIO14 中断 |
| IRQN_GPIOHS15_INTERRUPT | 高速 GPIO15 中断 |
| IRQN_GPIOHS16_INTERRUPT | 高速 GPIO16 中断 |
| IRQN_GPIOHS17_INTERRUPT | 高速 GPIO17 中断 |
| IRQN_GPIOHS18_INTERRUPT | 高速 GPIO18 中断 |
| IRQN_GPIOHS19_INTERRUPT | 高速 GPIO19 中断 |
| IRQN_GPIOHS20_INTERRUPT | 高速 GPIO20 中断 |
| IRQN_GPIOHS21_INTERRUPT | 高速 GPIO21 中断 |
| IRQN_GPIOHS22_INTERRUPT | 高速 GPIO22 中断 |
| IRQN_GPIOHS23_INTERRUPT | 高速 GPIO23 中断 |
| IRQN_GPIOHS24_INTERRUPT | 高速 GPIO24 中断 |
| IRQN_GPIOHS25_INTERRUPT | 高速 GPIO25 中断 |
| IRQN_GPIOHS26_INTERRUPT | 高速 GPIO26 中断 |
| IRQN_GPIOHS27_INTERRUPT | 高速 GPIO27 中断 |
| IRQN_GPIOHS28_INTERRUPT | 高速 GPIO28 中断 |
| IRQN_GPIOHS29_INTERRUPT | 高速 GPIO29 中断 |
| IRQN_GPIOHS30_INTERRUPT | 高速 GPIO30 中断 |
| IRQN_GPIOHS31_INTERRUPT | 高速 GPIO31 中断 |

## plic_irq_callback_t

### 描述

外部中断回调函数。

### 定义

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