3. 8keypad 控制 RGB 灯

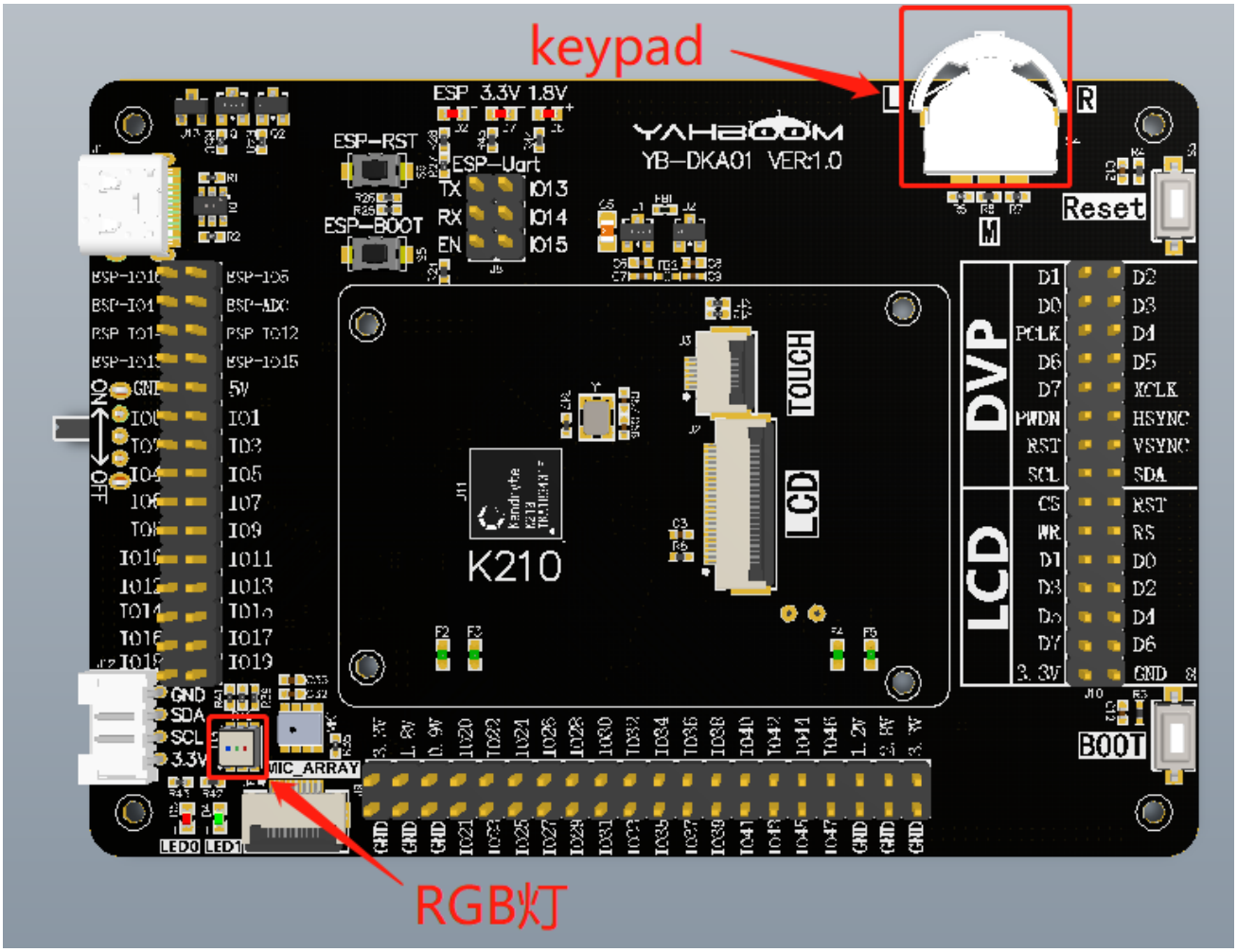
一、实验目的

本节课主要学习 K210 的拨轮开关 keypad 控制 RGB 灯。

二、实验准备

1. 实验元件

拨轮开关 keypad、RGB 灯



2. 元件特性

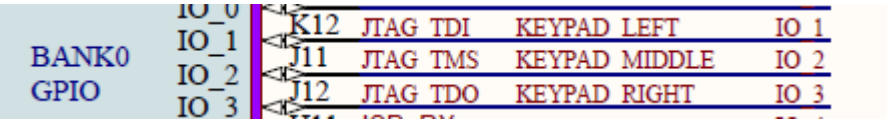
拨轮开关 keypad 具有三个通道，分别是 L：表示向左滚动、M：表示按下，R：表示向右滚动。每一次只能操作一个通道，并且 keypad 在释放状态下，三个通

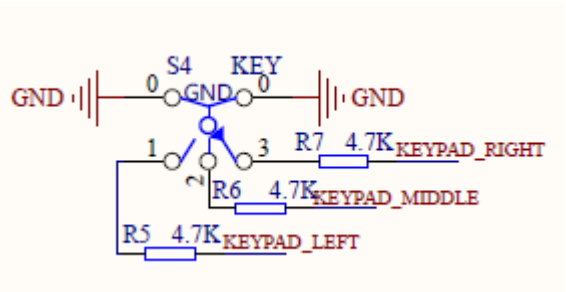
道都是高电平，如果其中一个通道被按下时，对应的 IO 口电平会变成低电平。可以理解为 keypad 是三个按键组合在一起，但是每次只能使用其中一个按键。

拨轮开关一般用于低压电路，具有滑块动作灵活、性能稳定可靠的特点。主要用于工业以及一些家用电器上，如电视机，彩电、电脑、数控机床、主动化操控设备等运用。

3.硬件连接

K210 开发板出厂默认已经焊接好 RGB 灯和拨轮开关 keypad。RGB 灯 R 连接的是 IO6，G 连接的是 IO7，B 连接的是 IO8。keypad_left 连接的是 IO1，keypad_middle 连接的是 IO2，keypad_right 连接的是 IO3。





4.SDK 中对应 API 功能

对应的头文件 gpiohs.h

前面已经介绍过 gpiohs 中的函数功能，这里就不再赘述。

三、实验原理

拨轮开关 keypad 是通过拨动开关柄使电路接通或断开，从而达到切换电路的目的，拨轮开关的原理是经过人为的操作，控制对应电路接通，这里的作用是

接通 GND，使 IO 口电平变为低电平，松开时弹簧自动复位的过程。

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。

```
/*****************************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_KEYPAD_LEFT         (1)
#define PIN_KEYPAD_MIDDLE       (2)
#define PIN_KEYPAD_RIGHT        (3)

/*****************************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 KEYPAD_LEFT_GPIONUM     (3)
#define KEYPAD_MIDDLE_GPIONUM   (4)
#define KEYPAD_RIGHT_GPIONUM    (5)

/*****************************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_KEYPAD_LEFT        (FUNC_GPIOHS0 + KEYPAD_LEFT_GPIONUM)
#define FUNC_KEYPAD_MIDDLE      (FUNC_GPIOHS0 + KEYPAD_MIDDLE_GPIONUM)
#define FUNC_KEYPAD_RIGHT       (FUNC_GPIOHS0 + KEYPAD_RIGHT_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_KEYPAD_LEFT,   FUNC_KEYPAD_LEFT);
    fpioa_set_function(PIN_KEYPAD_MIDDLE, FUNC_KEYPAD_MIDDLE);
    fpioa_set_function(PIN_KEYPAD_RIGHT,  FUNC_KEYPAD_RIGHT);

}
```

2. 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();
}
```

3. 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);
}
```

4. 初始化 keypad，设置 GPIO 为上拉输入模式。

```
void init_keypad(void)
{
    /* 设置keypad的GPIO模式为上拉输入 */
    gpiohs_set_drive_mode(KEYPAD_LEFT_GPIONUM,   GPIO_DM_INPUT_PULL_UP);
    gpiohs_set_drive_mode(KEYPAD_MIDDLE_GPIONUM, GPIO_DM_INPUT_PULL_UP);
    gpiohs_set_drive_mode(KEYPAD_RIGHT_GPIONUM,  GPIO_DM_INPUT_PULL_UP);
}
```

5. 接下来是扫描 keypad 的状态，先读取 keypad 三个通道的 GPIO 状态，然后检测是否向左滚动，如果是则让 RGB 亮红灯，其中 msleep(10)延迟 10 毫秒是起到消抖的作用。之所以要消抖，是因为按键在按下或者松开的时候，电平从高变低或者从低变高的过程不是立即变化的，而是有变化过程的，所以需要消抖处理。

```
void scan_keypad(void)
{
    /* 读取keypad三个通道的状态 */
    gpio_pin_value_t state_keypad_left =   gpiohs_get_pin(KEYPAD_LEFT_GPIONUM);
    gpio_pin_value_t state_keypad_middle = gpiohs_get_pin(KEYPAD_MIDDLE_GPIONUM);
    gpio_pin_value_t state_keypad_right =  gpiohs_get_pin(KEYPAD_RIGHT_GPIONUM);

    /* 检测keypad是否向左滚动 */
    if (!state_keypad_left)
    {
        /* 延迟消抖10ms */
        msleep(10);
        /* 再次读keypad向左的IO口的状态 */
        state_keypad_left = gpiohs_get_pin(KEYPAD_LEFT_GPIONUM);
        if (!state_keypad_left)
        {
            /* 向左滚动，点亮红灯 */
            gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_LOW);
        }
        else
        {
            /* 松开，红灯熄灭 */
            gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
        }
    }
```

6. 检测 keypad 是否被按下，如果是则 RGB 点亮绿灯，否则绿灯熄灭。

```
/* 检测keypad是否被按下 */
else if (!state_keypad_middle)
{
    msleep(10);
    state_keypad_middle = gpiohs_get_pin(KEYPAD_MIDDLE_GPIONUM);
    if (!state_keypad_middle)
    {
        gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_LOW);
    }
    else
    {
        gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
    }
}
```

7. 检测 keypad 是否向右滚动，如果是则 RGB 点亮蓝灯，否则蓝灯熄灭。

```
/* 检测keypad是否向右滚动 */
else if (!state_keypad_right)
{
    msleep(10);
    state_keypad_right = gpiohs_get_pin(KEYPAD_RIGHT_GPIONUM);
    if (!state_keypad_right)
    {
        gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_LOW);
    }
    else
    {
        gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
    }
}
```

8. 最后是一个 while(1)循环，扫描 keypad 的状态控制 RGB 灯。

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

    /* 初始化RGB灯 */
    init_rgb();

    /* 初始化keypad */
    init_keypad();

    while (1)
    {
        /* 扫描keypad并控制RGB灯 */
        scan_keypad();
    }

    return 0;
}
```

9.编译调试，烧录运行

把本课程资料中的 keypad 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=keypad -G "MinGW Makefiles"

make

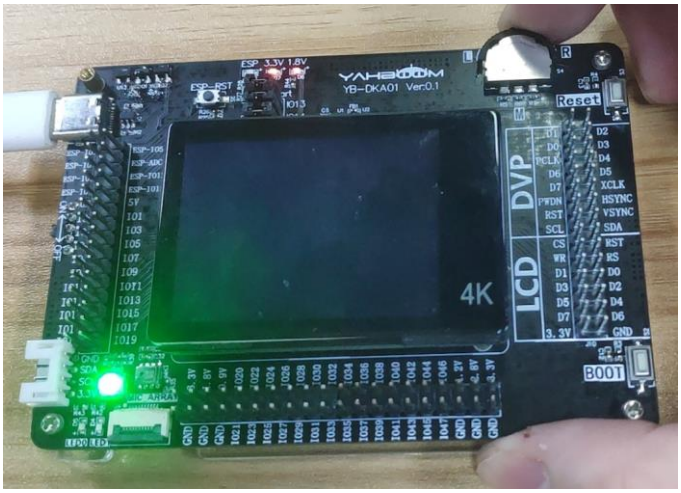
```
Scanning dependencies of target keypad
[ 97%] Building C object CMakeFiles/keypad.dir/src/keypad/main.c.obj
[100%] Linking C executable keypad
Generating .bin file ...
[100%] Built target keypad
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

编译完成后，在 build 文件夹下会生成 keypad.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

当拨动 keypad 向左滚动时，RGB 亮红灯，松开则熄灭；当拨动 keypad 向右滚动时，RGB 亮蓝灯，松开同样熄灭；当按下 keypad 时，RGB 亮绿灯，松开同样熄灭。每次只能控制 RGB 亮一种颜色，不能亮多种颜色。



六、实验总结

1.keypad 的内部原理其实是三个按键，只不过同一时间只能触发一个按键按下。

2.keypad 读取 GPIO 电平的方法与按键是一样的,所以它也支持中断处理的方式。

3.keypad 操作简单，具有弹簧复位的功能，实际操作很方便。

附：API 对应的头文件 gpiohs.h

gpiohs_set_drive_mode

描述

设置 GPIO 驱动模式。

函数原型

```
void gpiohs_set_drive_mode(uint8_t pin, gpio_drive_mode_t mode)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
mode	GPIO 驱动模式	输入

返回值

无。

gpio_set_pin

描述

设置 GPIO 管脚值。

函数原型

```
void gpiohs_set_pin(uint8_t pin, gpio_pin_value_t value)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
value	GPIO 值	输入

返回值

无。

gpio_get_pin

描述

获取 GPIO 管脚值。

函数原型

```
gpio_pin_value_t gpiohs_get_pin(uint8_t pin)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入

返回值

获取的 GPIO 管脚值。

gpiohs_set_pin_edge

描述

设置高速 GPIO 中断触发模式。

函数原型

```
void gpiohs_set_pin_edge(uint8_t pin, gpio_pin_edge_t edge)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
edge	中断触发方式	输入

返回值

无。

gpiohs_set_irq

描述

设置高速 GPIO 的中断回调函数。

函数原型

```
void gpiohs_set_irq(uint8_t pin, uint32_t priority, void(*func)());
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
priority	中断优先级	输入
func	中断回调函数	输入

返回值

无。

gpiohs_irq_register

描述

设置高速 GPIO 的中断回调函数。

函数原型

```
void gpiohs_irq_register(uint8_t pin, uint32_t priority, plic_irq_callback_t
callback, void *ctx)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
priority	中断优先级	输入
plic_irq_callback_t	中断回调函数	输入
ctx	回调函数参数	输入

返回值

无。

gpiohs_irq_unregister

描述

注销 GPIOHS 中断。

函数原型

```
void gpiohs_irq_unregister(uint8_t pin)
```

参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入

返回值

无。

数据类型

相关数据类型、数据结构定义如下：

- gpio_drive_mode_t：GPIO 驱动模式。
- gpio_pin_value_t：GPIO 值。
- gpio_pin_edge_t：GPIO 边沿触发模式。

gpio_drive_mode_t

描述

GPIO 驱动模式。

定义

```
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;
```

成员

成员名称	描述
GPIO_DM_INPUT	输入
GPIO_DM_INPUT_PULL_DOWN	输入下拉
GPIO_DM_INPUT_PULL_UP	输入上拉
GPIO_DM_OUTPUT	输出

gpio_pin_value_t

描述

GPIO 值。

定义

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

成员

成员名称	描述
GPIO_PV_LOW	低
GPIO_PV_HIGH	高

gpio_pin_edge_t

描述

高速 GPIO 边沿触发模式。

定义

```
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;
```

成员

成员名称	描述
GPIO_PE_NONE	不触发
GPIO_PE_FALLING	下降沿触发
GPIO_PE_RISING	上升沿触发
GPIO_PE_BOTH	双沿触发
GPIO_PE_LOW	低电平触发
GPIO_PE_HIGH	高电平触发