3.8 Keypad control RGB

1. Experiment purpose

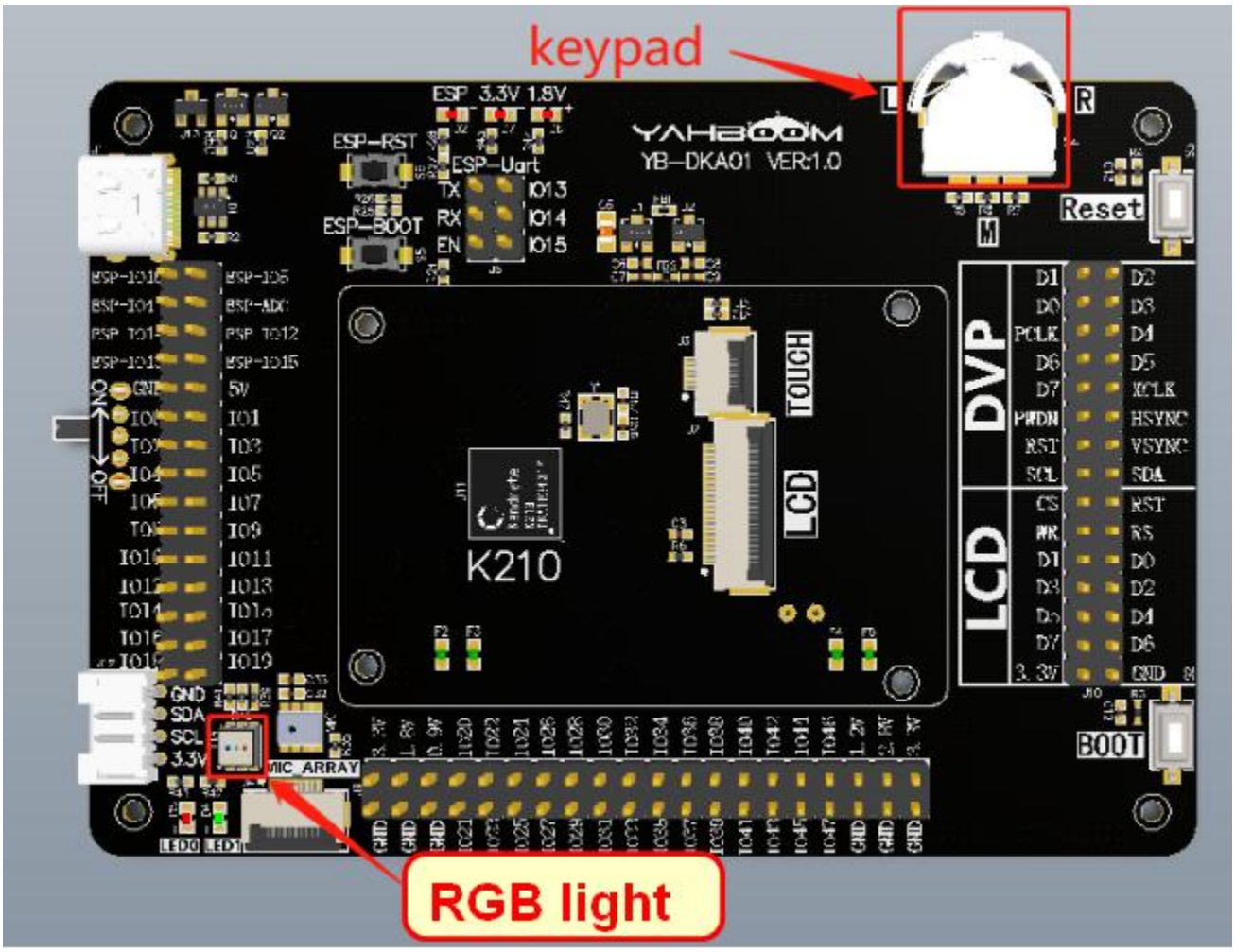
In this lesson, we mainly learns how to use K210's dial switch keypad to control RGB lights.

2.Experiment preparation

2.1 components

Separate button, BOOT LED light

dial switch keypad, RGB light



2.2 Component characteristics

The dial switch keypad possess three channels.

L-scroll to the left,

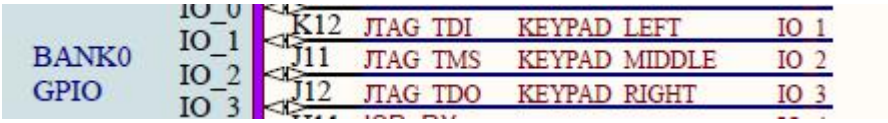
M-press,

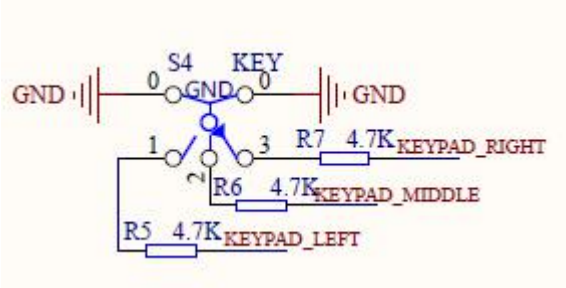
R-scroll to the right.

Only one channel can be operated at a time, and when the keypad is in the released state, the three channels are high. If one of the channels is pressed, the corresponding IO port level will become low.

2.3 Hardware connection

The K210 development board has already been welded with the RGB light and the dial switch keypad by default. RGB light R is connected to IO6, G is connected to IO7, and B is connected to IO8. The keypad_left is connected to IO1, the keypad_middle is connected to IO2, and the keypad_right is connected to IO3.





2.4 SDK API function

The header file is gpiohs.h

3. Experimental principle

The dial switch keypad is to switch the circuit on or off by turning the switch handle, so as to achieve the purpose of switching the circuit. The function in K210 board is to connect GND make the IO port level becomes low and the spring automatically resets when released.

4. Experiment procedure

4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.

```
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#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********************************/
//Software GPIO port, corresponding program
#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************************************/
//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_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 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_KEYPAD_LEFT,   FUNC_KEYPAD_LEFT);
    fpioa_set_function(PIN_KEYPAD_MIDDLE, FUNC_KEYPAD_MIDDLE);
    fpioa_set_function(PIN_KEYPAD_RIGHT,  FUNC_KEYPAD_RIGHT);

}
```

4.2 It needs to be initialized before using RGB light, that is, set the software GPIO of RGB light to 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();
}
```

4.3 Set the GPIO of the RGB light to high to make the RGB light go out.

```
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 Initialized keypad light, that is, set the software GPIO of RGB light to Pull_UP input mode.

```
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);
}
```

4.5 Scanning the state of the keypad.

First read the GPIO state of the three channels of the keypad. Then, check whether it is scrolling to the left. If it is, make the RGB light red, msleep(10) be used to debounce.

```
void scan_keypad(void)
{
    /*Read the status of the three channels of the keypad*/
    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);

    /*Check if the keypad is scrolling to the left*/
    if (!state_keypad_left)
    {
        /*Delay debounce 10ms*/
        msleep(10);
        /*Read the status of the IO port to the left of the keypad again*/
        state_keypad_left = gpiohs_get_pin(KEYPAD_LEFT_GPIONUM);
        if (!state_keypad_left)
        {
            /*Scroll to the left to light up the red light */
            gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_LOW);
        }
        else
        {
            /*Release, the red light is off*/
            gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
        }
    }
```

4.6 Check whether the keypad is pressed, if it is, the RGB light will become green, otherwise the green light will go out.

```
    /* Check whether the keypad is pressed */
    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);
        }
    }
```

4.7 Check whether it is scrolling to the right, if it is, the RGB light will become blue, otherwise the green light will go out.

```
/* Check if the keypad is scrolling to the right*/
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);
    }
}
```

4.8 The last is a while(1) loop, which scans the state of the keypad to control the RGB lights.

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

    /* Initialize RGB lights*/
    init_rgb();

    /*  Initialize keypad */
    init_keypad();

    while (1)
    {
        /* Scan the keypad and control the RGB lights*/
        scan_keypad();
    }

    return 0;
}
```

4.9 Compile and debug, burn and run

Copy the keypad to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

cmake .. -DPROJ=keypad -G "MinGW Makefiles"

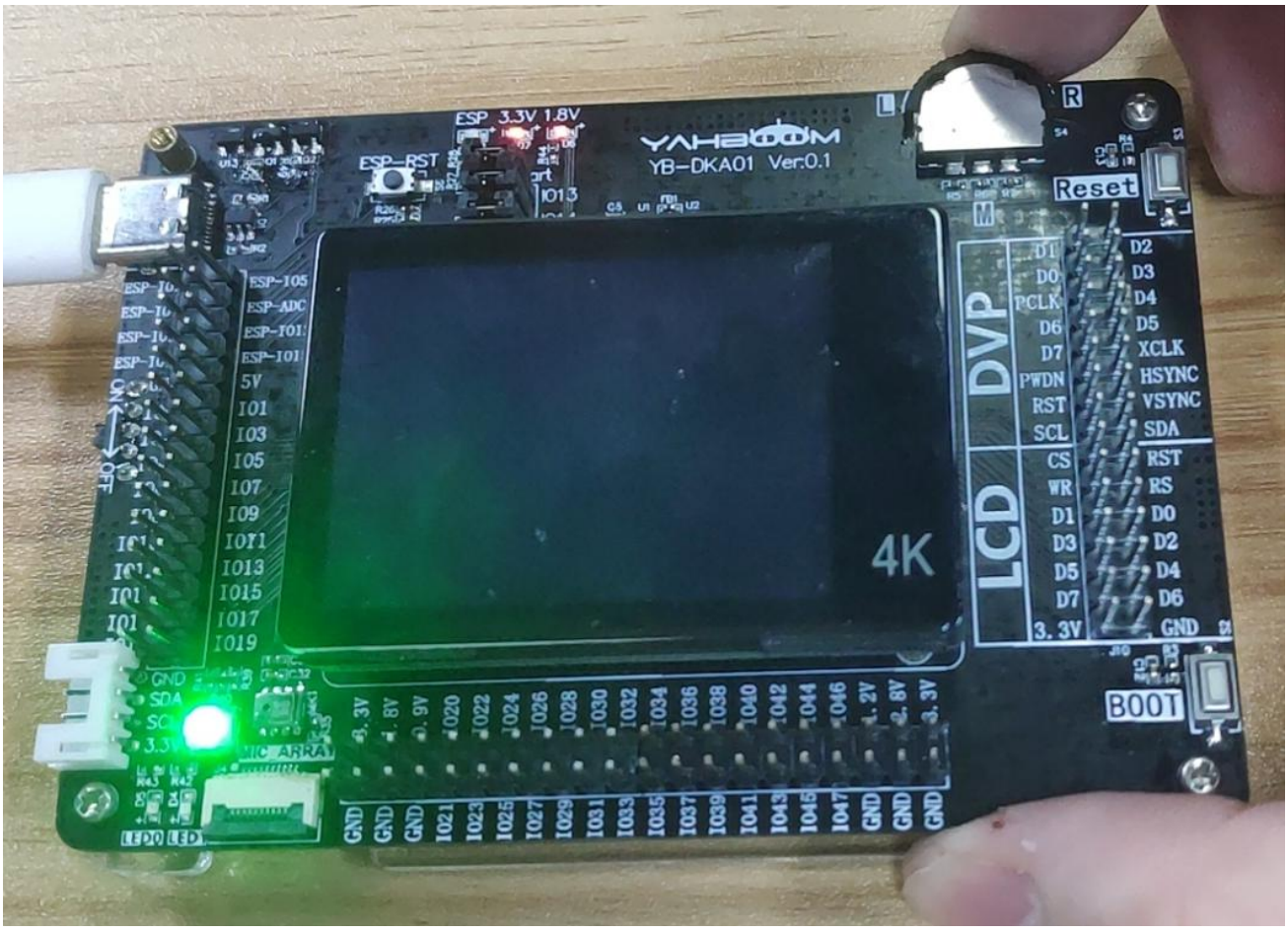
make

```
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>
```

After the compilation is complete, the timer.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

When the keypad is scrolled to the left, RGB lights become red.
When the keypad is scrolled to the right, RGB lights become blue.
When the keypad is released, it lights up in green.



6. Experiment summary

1. The internal principle of the keyboard is actually three buttons, only one button can be triggered at the same time.
2. The method for keypad to read GPIO level is the same as that for keys, it also supports interrupt handling.

3. The keypad is easy to operate, and it has a spring reset function.

Appendix -- API

Header file is gpiohs.h

gpiohs_set_drive_mode

Description: Set GPIO 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

gpio_set_pin

Description: Set GPIO 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

gpio_get_pin

Description: Get GPIO value.

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

Parameter:

Parameter name	Description	Input/Output
pin	GPIO	Input

Return value: Get GPIO value

gpiohs_set_pin_edge

Description: Set the high-speed GPIO 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

gpiohs_set_irq

Description: Set the interrupt callback function of high-speed GPIO.

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

gpiohs_irq_register

Description: Set the interrupt callback function of high-speed GPIO.

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	GPIO	Input
priority	Interrupt priority	Input
plic_irq_callback_t	Interrupt callback function	Input
ctx	Callback function	Input

Return value: No

gpiohs_irq_unregister

Description: 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

Data type

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

gpio_drive_mode_t

Description

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

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

gpio_pin_edge_t

Description

High-speed GPIO 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	Falling edge trigger

Member name	Description
GPIO_PE_RISING	Rising edge trigger
GPIO_PE_BOTH	Double edge trigger
GPIO_PE_LOW	Low trigger
GPIO_PE_HIGH	High trigger