5.1 Keypad state machine events

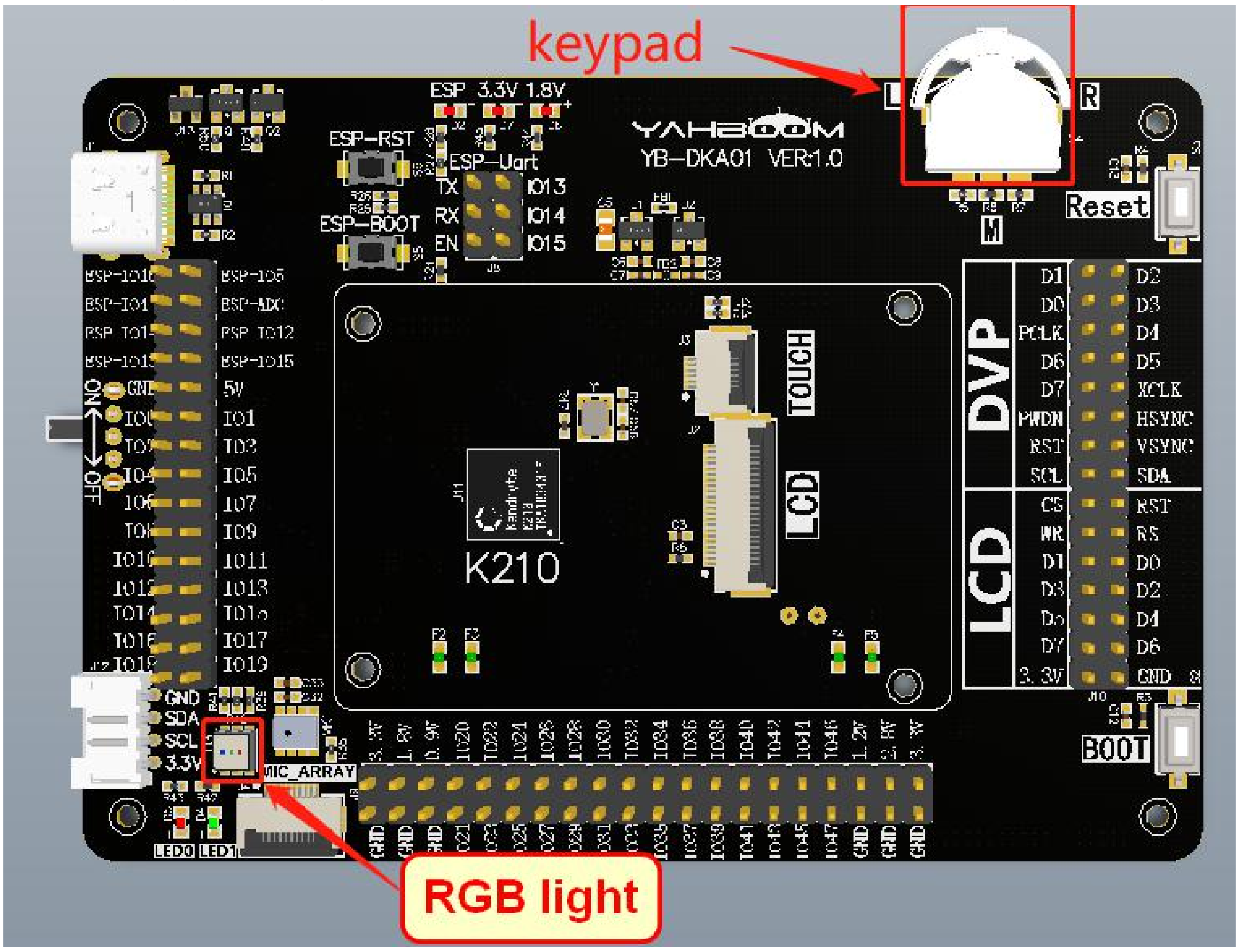
1. Experiment purpose

In this lesson, we mainly learn K210's dial switch keypad uses a state machine to detect key events.

2.Experiment preparation

2.1 components

dial switch keypad, RGB light



3. Experiment principle

The dial switch keypad is to switch the circuit on or off by turning the switch handle.

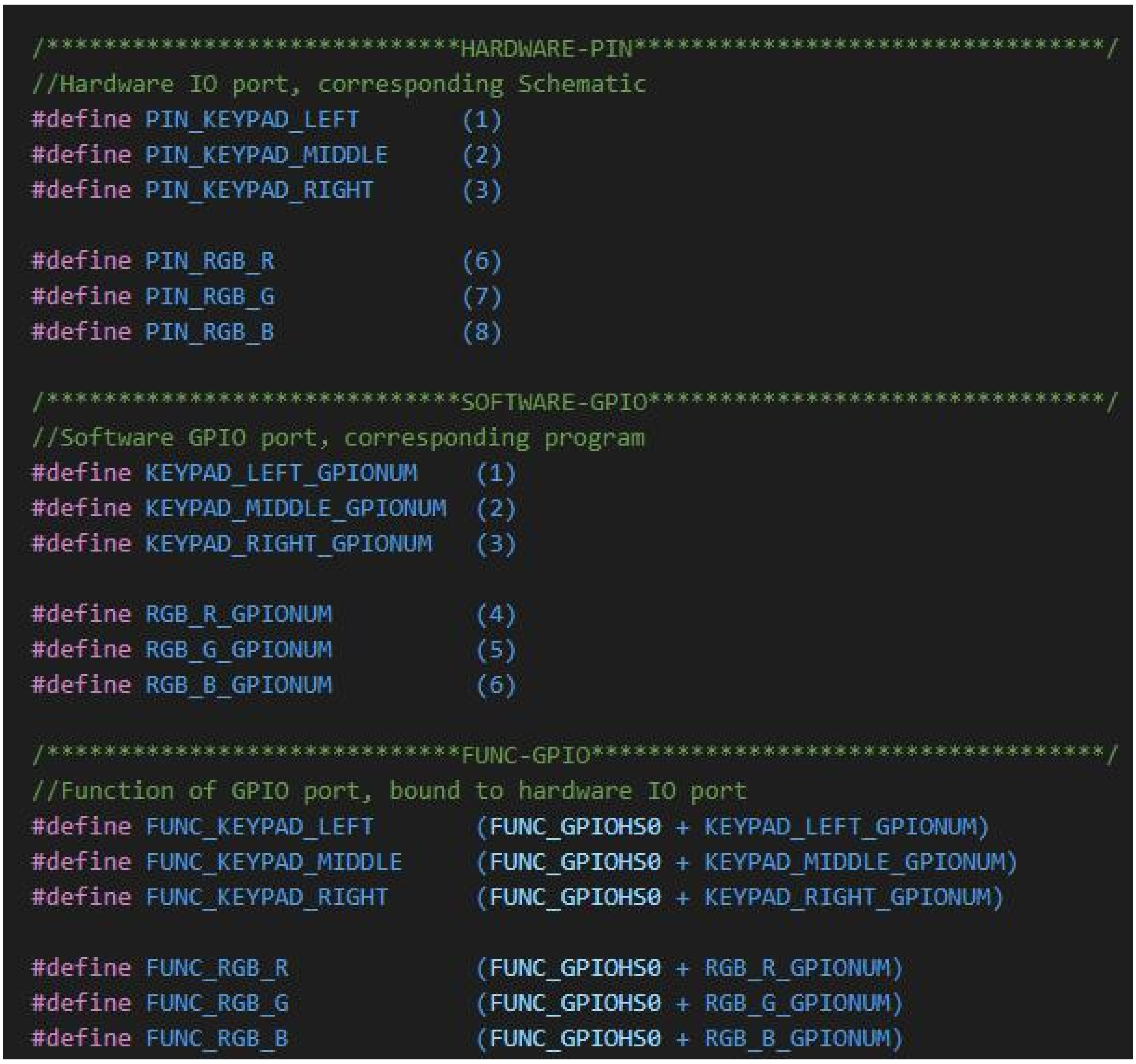
The principle of the dial switch is that control the corresponding circuit to be turned on through artificial operation. The function is to turn on the GND, so that the IO port level becomes low, and the spring automatically resets when it is released.

In programming, we can use the timer to scan the state of the keypad, calculate the time value of the keypad state during the timer scan process, thereby read the key state of the three channels of the keypad, store it in the FIFO queue mechanism, and execute each state through the function callback.

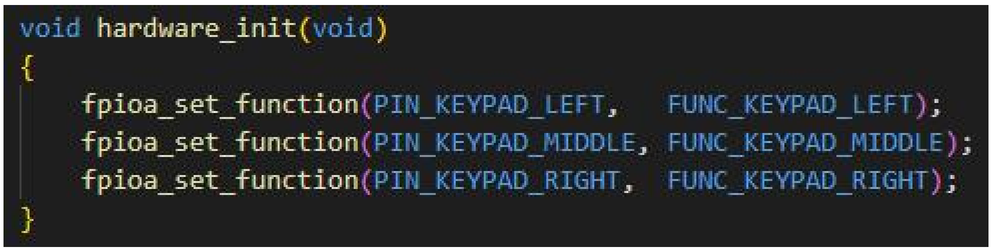
Or we can read the state of the button, and then process the event according to the state.

4. Experiment procedure

4.1 K210's hardware pins and software functions use the FPIOA mapping relationship.

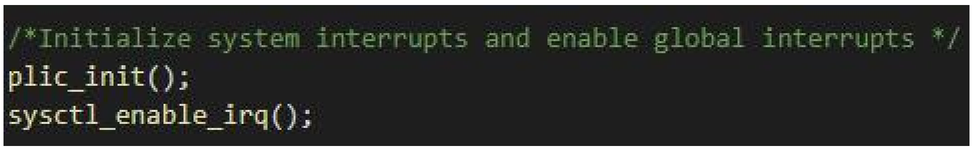


```c
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_KEYPAD_LEFT       (1)
#define PIN_KEYPAD_MIDDLE     (2)
#define PIN_KEYPAD_RIGHT      (3)

#define PIN_RGB_R             (6)
#define PIN_RGB_G             (7)
#define PIN_RGB_B             (8)

/*****************************SOFTWARE-GPIO********************************/
//Software GPIO port, corresponding program
#define KEYPAD_LEFT_GPIONUM    (1)
#define KEYPAD_MIDDLE_GPIONUM  (2)
#define KEYPAD_RIGHT_GPIONUM   (3)

#define RGB_R_GPIONUM          (4)
#define RGB_G_GPIONUM          (5)
#define RGB_B_GPIONUM          (6)

/*****************************FUNC-GPIO************************************/
//Function of GPIO port, bound to hardware IO port
#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)

#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)
```



```c
void hardware_init(void)
{
    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 Due to the need to use the timer interrupt, it is necessary to initialize the system interrupt and enable the global interrupt.



```c
/*Initialize system interrupts and enable global interrupts */
plic_init();
sysctl_enable_irq();
```

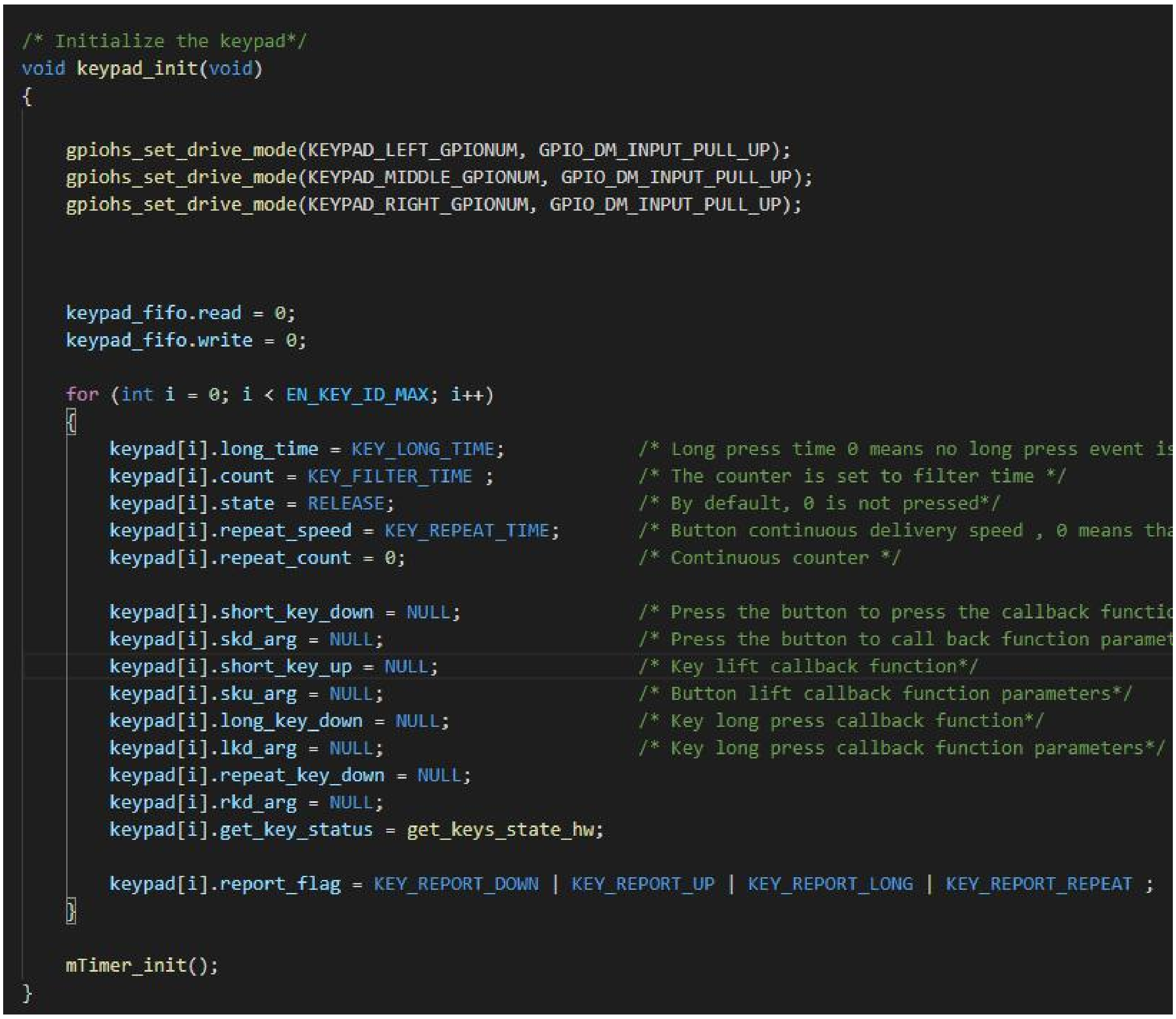
4.3 It needs to be initialized before using RGB lights.

```c
/* Initialize the RGB */
rgb_init(EN_RGB_ALL);
```

```c
/* Initialize the RGB lamp */
void rgb_init(rgb_color_t color)
{
    switch (color)
    {
    case EN_RGB_RED:
        fpioa_set_function(PIN_RGB_R, FUNC_RGB_R);
        gpiohs_set_drive_mode(RGB_R_GPIONUM, GPIO_DM_OUTPUT);
        gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
        break;
    case EN_RGB_GREEN:
        fpioa_set_function(PIN_RGB_G, FUNC_RGB_G);
        gpiohs_set_drive_mode(RGB_G_GPIONUM, GPIO_DM_OUTPUT);
        gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
        break;
    case EN_RGB_BLUE:
        fpioa_set_function(PIN_RGB_B, FUNC_RGB_B);
        gpiohs_set_drive_mode(RGB_B_GPIONUM, GPIO_DM_OUTPUT);
        gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
        break;
    case EN_RGB_ALL:
        fpioa_set_function(PIN_RGB_R, FUNC_RGB_R);
        gpiohs_set_drive_mode(RGB_R_GPIONUM, GPIO_DM_OUTPUT);
        fpioa_set_function(PIN_RGB_G, FUNC_RGB_G);
        gpiohs_set_drive_mode(RGB_G_GPIONUM, GPIO_DM_OUTPUT);
        fpioa_set_function(PIN_RGB_B, FUNC_RGB_B);
        gpiohs_set_drive_mode(RGB_B_GPIONUM, GPIO_DM_OUTPUT);

        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);
        break;

    default:
        break;
    }
}
```

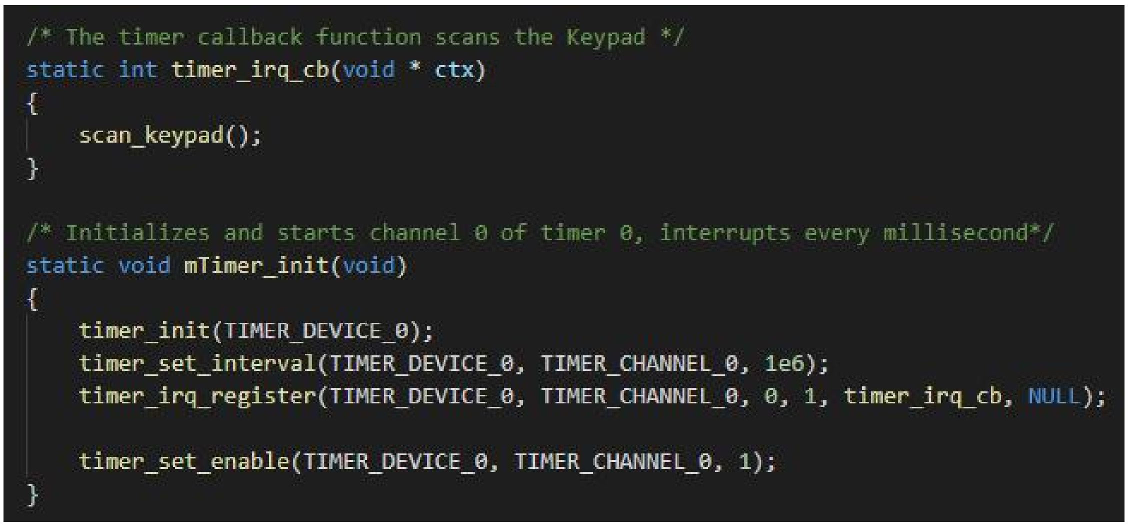
4.4 Initialize the keypad, set the GPIO to pull-up input mode, clear the FIFO. Then, set the initial data of the three channels of the keypad, and finally initialize and start the timer.

```c
/* Initialize the keypad */
keypad_init();
```



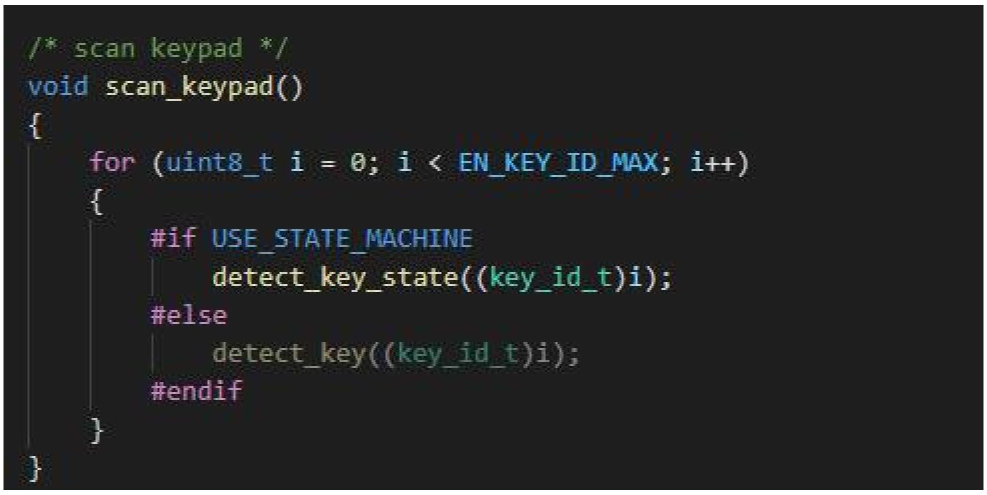
```c
/* Initialize the keypad*/
void keypad_init(void)
{

    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);



    keypad_fifo.read = 0;
    keypad_fifo.write = 0;

    for (int i = 0; i < EN_KEY_ID_MAX; i++)
    {
        keypad[i].long_time = KEY_LONG_TIME;            /* Long press time 0 means no long press event is
        keypad[i].count = KEY_FILTER_TIME ;             /* The counter is set to filter time */
        keypad[i].state = RELEASE;                      /* By default, 0 is not pressed*/
        keypad[i].repeat_speed = KEY_REPEAT_TIME;       /* Button continuous delivery speed , 0 means tha
        keypad[i].repeat_count = 0;                     /* Continuous counter */

        keypad[i].short_key_down = NULL;                /* Press the button to press the callback functio
        keypad[i].skd_arg = NULL;                       /* Press the button to call back function paramet
        keypad[i].short_key_up = NULL;                  /* Key lift callback function*/
        keypad[i].sku_arg = NULL;                       /* Button lift callback function parameters*/
        keypad[i].long_key_down = NULL;                 /* Key long press callback function*/
        keypad[i].lkd_arg = NULL;                       /* Key long press callback function parameters*/
        keypad[i].repeat_key_down = NULL;
        keypad[i].rkd_arg = NULL;
        keypad[i].get_key_status = get_keys_state_hw;

        keypad[i].report_flag = KEY_REPORT_DOWN | KEY_REPORT_UP | KEY_REPORT_LONG | KEY_REPORT_REPEAT ;
    }

    mTimer_init();
}
```

4.5 Using channel 0 of timer 0, and the timer interrupt time is 1 millisecond, that is, the keypad is scanned every millisecond.



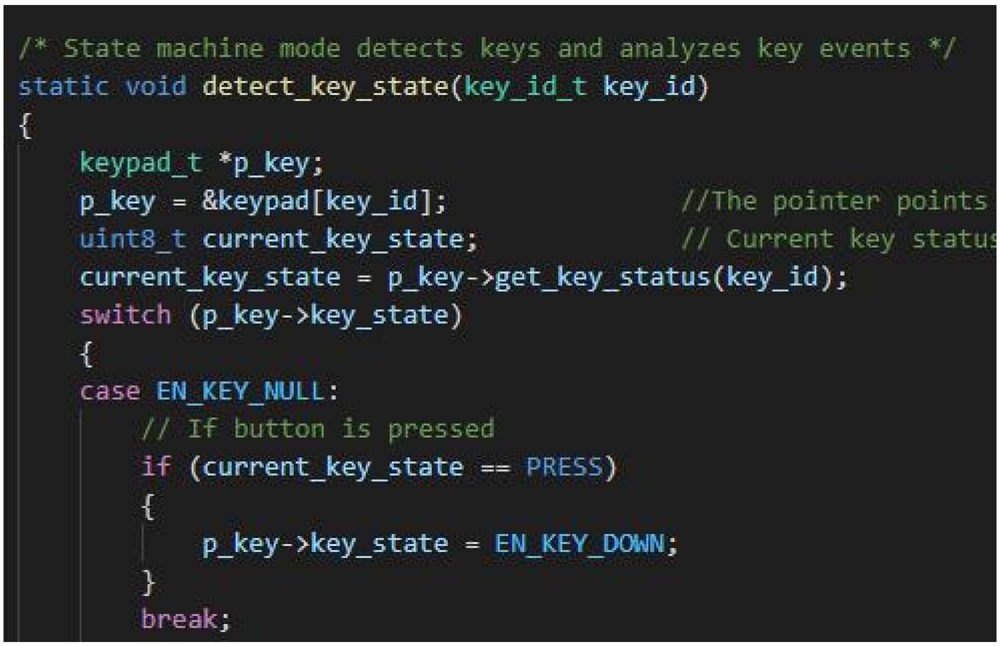
```c
/* The timer callback function scans the Keypad */
static int timer_irq_cb(void * ctx)
{
    scan_keypad();
}

/* Initializes and starts channel 0 of timer 0, interrupts every millisecond*/
static void mTimer_init(void)
{
    timer_init(TIMER_DEVICE_0);
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1e6);
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_irq_cb, NULL);

    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

4.6 There are two ways to choose in the scan keypad function, the default is the state machine scan.



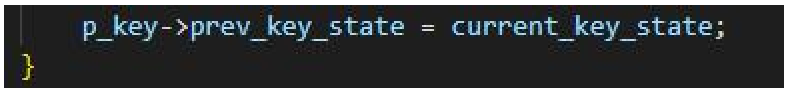
```
/* scan keypad */
void scan_keypad()
{
    for (uint8_t i = 0; i < EN_KEY_ID_MAX; i++)
    {
        #if USE_STATE_MACHINE
            detect_key_state((key_id_t)i);
        #else
            detect_key((key_id_t)i);
        #endif
    }
}
```

4.7 Due to the nature of keypad, only one channel is valid at a time.
We need to get the ID of the corresponding channel and only process the valid ID.



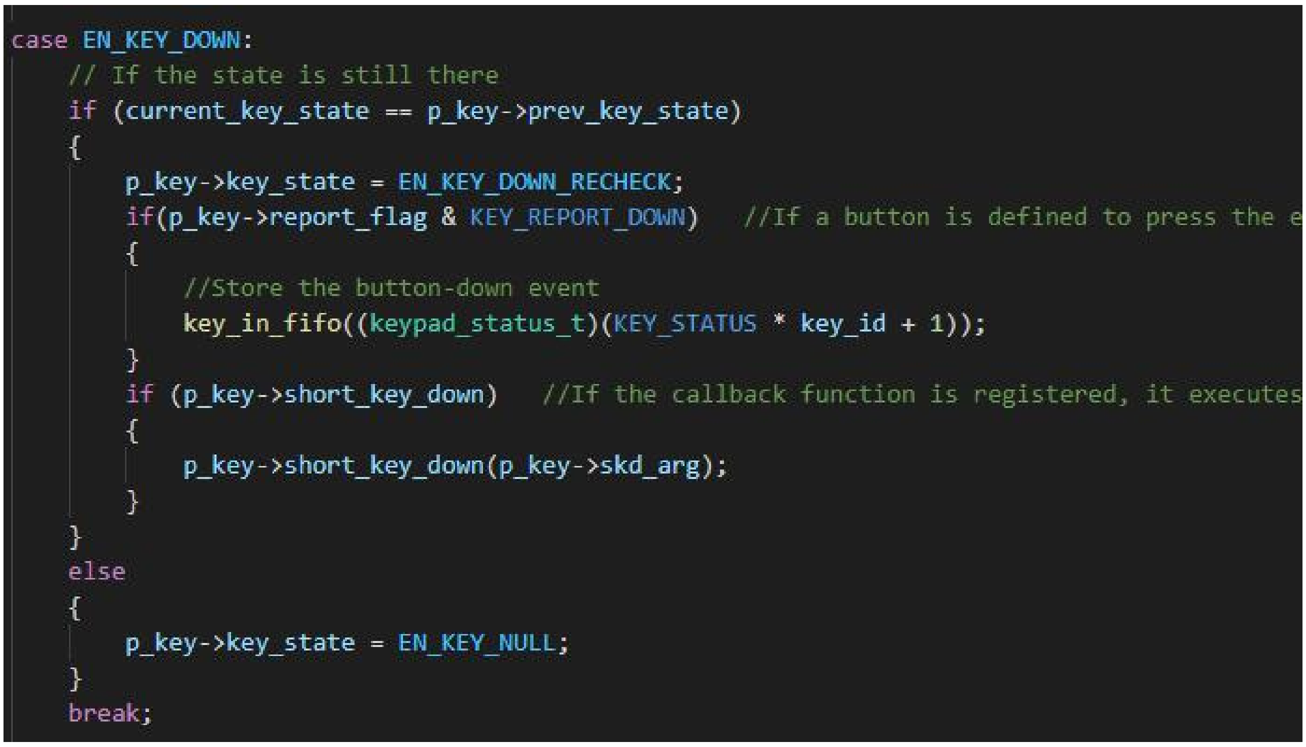
```
/* State machine mode detects keys and analyzes key events */
static void detect_key_state(key_id_t key_id)
{
    keypad_t *p_key;
    p_key = &keypad[key_id];                 //The pointer points
    uint8_t current_key_state;               // Current key status
    current_key_state = p_key->get_key_status(key_id);
    switch (p_key->key_state)
    {
    case EN_KEY_NULL:
        // If button is pressed
        if (current_key_state == PRESS)
        {
            p_key->key_state = EN_KEY_DOWN;
        }
        break;
```

At the end of each scan, the current state will be passed to prev_key_state, indicating the previous state.



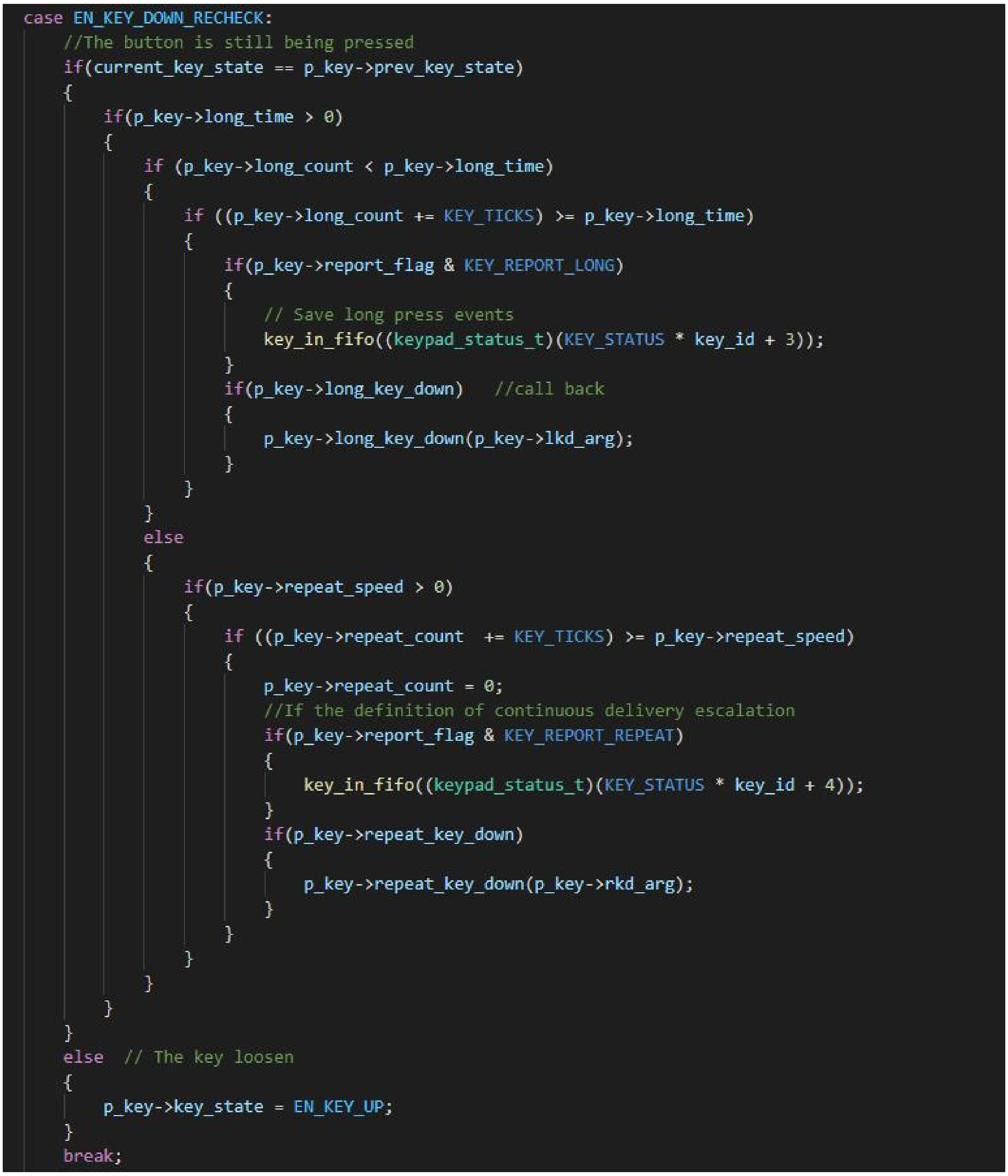
```
    p_key->prev_key_state = current_key_state;
}
```

In the EN_KEY_DOWN state, it will first judge whether the changed state is maintained, which is equivalent to the debounce function.
Then, report the key press event, store it in the FIFO, execute the callback function, and modify the state to EN_KEY_DOWN_RECHECK.



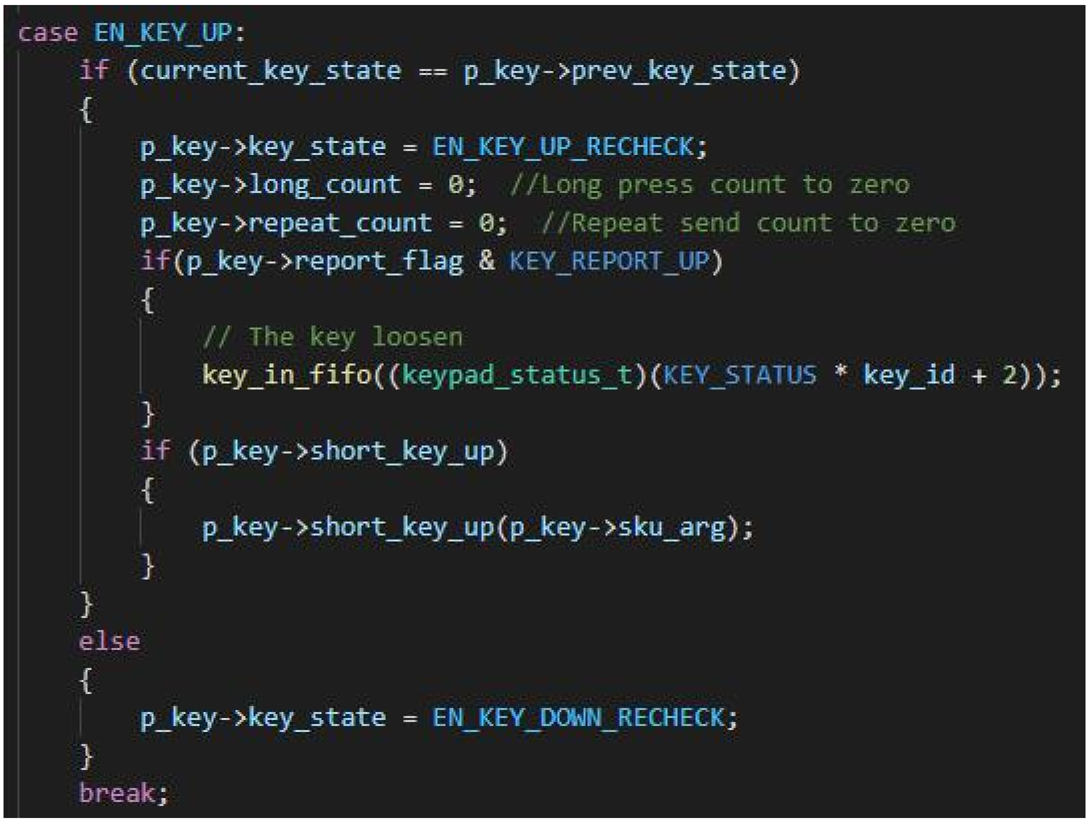
```
case EN_KEY_DOWN:
    // If the state is still there
    if (current_key_state == p_key->prev_key_state)
    {
        p_key->key_state = EN_KEY_DOWN_RECHECK;
        if(p_key->report_flag & KEY_REPORT_DOWN)   //If a button is defined to press the e
        {
            //Store the button-down event
            key_in_fifo((keypad_status_t)(KEY_STATUS * key_id + 1));
        }
        if (p_key->short_key_down)   //If the callback function is registered, it executes
        {
            p_key->short_key_down(p_key->skd_arg);
        }
    }
    else
    {
        p_key->key_state = EN_KEY_NULL;
    }
    break;
```

The EN_KEY_DOWN_RECHECK state mainly deals with long press, continuous press and release.



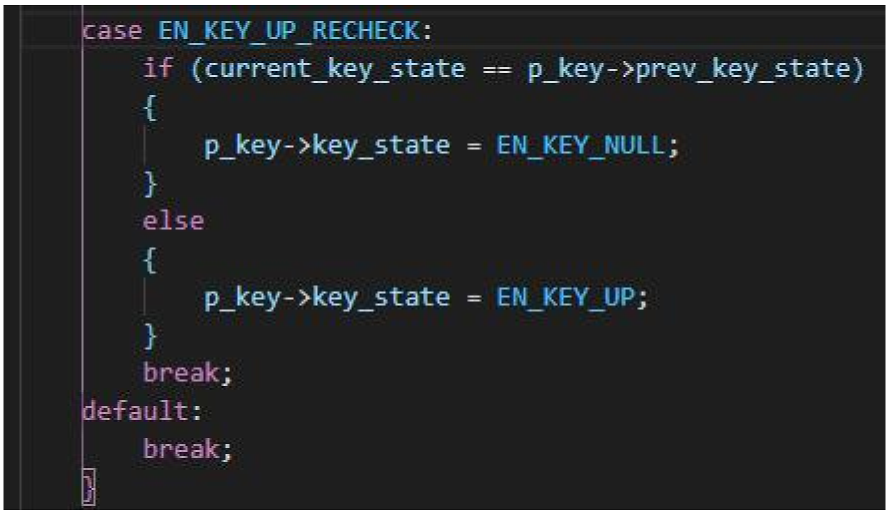
```
case EN_KEY_DOWN_RECHECK:
    //The button is still being pressed
    if(current_key_state == p_key->prev_key_state)
    {
        if(p_key->long_time > 0)
        {
            if (p_key->long_count < p_key->long_time)
            {
                if ((p_key->long_count += KEY_TICKS) >= p_key->long_time)
                {
                    if(p_key->report_flag & KEY_REPORT_LONG)
                    {
                        // Save long press events
                        key_in_fifo((keypad_status_t)(KEY_STATUS * key_id + 3));
                    }
                    if(p_key->long_key_down)   //call back
                    {
                        p_key->long_key_down(p_key->lkd_arg);
                    }
                }
            }
            else
            {
                if(p_key->repeat_speed > 0)
                {
                    if ((p_key->repeat_count  += KEY_TICKS) >= p_key->repeat_speed)
                    {
                        p_key->repeat_count = 0;
                        //If the definition of continuous delivery escalation
                        if(p_key->report_flag & KEY_REPORT_REPEAT)
                        {
                            key_in_fifo((keypad_status_t)(KEY_STATUS * key_id + 4));
                        }
                        if(p_key->repeat_key_down)
                        {
                            p_key->repeat_key_down(p_key->rkd_arg);
                        }
                    }
                }
            }
        }
    }
    else  // The key loosen
    {
        p_key->key_state = EN_KEY_UP;
    }
    break;
```

The EN_KEY_UP state will also first determine whether the values of the two states are the same. If they are the same, it means the button is released, and then trigger the release callback function, and the state is modified to EN_KEY_UP_RECHECK.



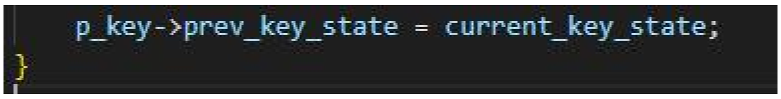
```
case EN_KEY_UP:
    if (current_key_state == p_key->prev_key_state)
    {
        p_key->key_state = EN_KEY_UP_RECHECK;
        p_key->long_count = 0;  //Long press count to zero
        p_key->repeat_count = 0;  //Repeat send count to zero
        if(p_key->report_flag & KEY_REPORT_UP)
        {
            // The key loosen
            key_in_fifo((keypad_status_t)(KEY_STATUS * key_id + 2));
        }
        if (p_key->short_key_up)
        {
            p_key->short_key_up(p_key->sku_arg);
        }
    }
    else
    {
        p_key->key_state = EN_KEY_DOWN_RECHECK;
    }
    break;
```

EN_KEY_UP_RECHECK is used to determine whether the key is released, and the modified state is EN_KEY_NULL.



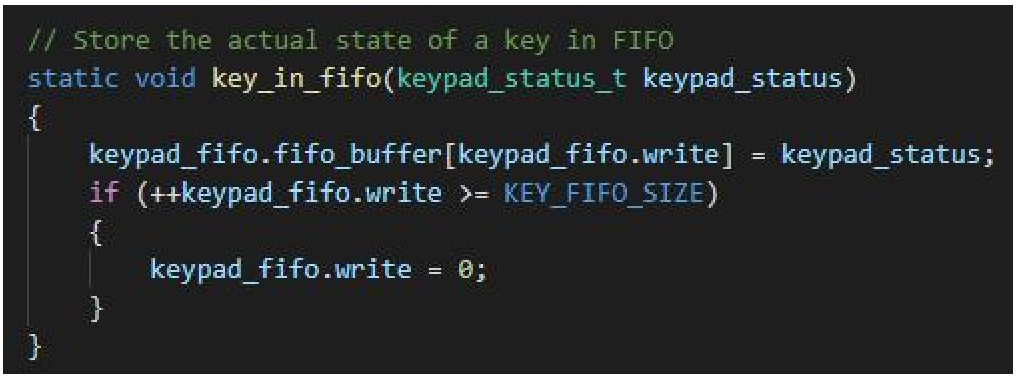
```
case EN_KEY_UP_RECHECK:
    if (current_key_state == p_key->prev_key_state)
    {
        p_key->key_state = EN_KEY_NULL;
    }
    else
    {
        p_key->key_state = EN_KEY_UP;
    }
    break;
default:
    break;
}
```

Save the current state to prev_key_state as the previous state of the next scan.



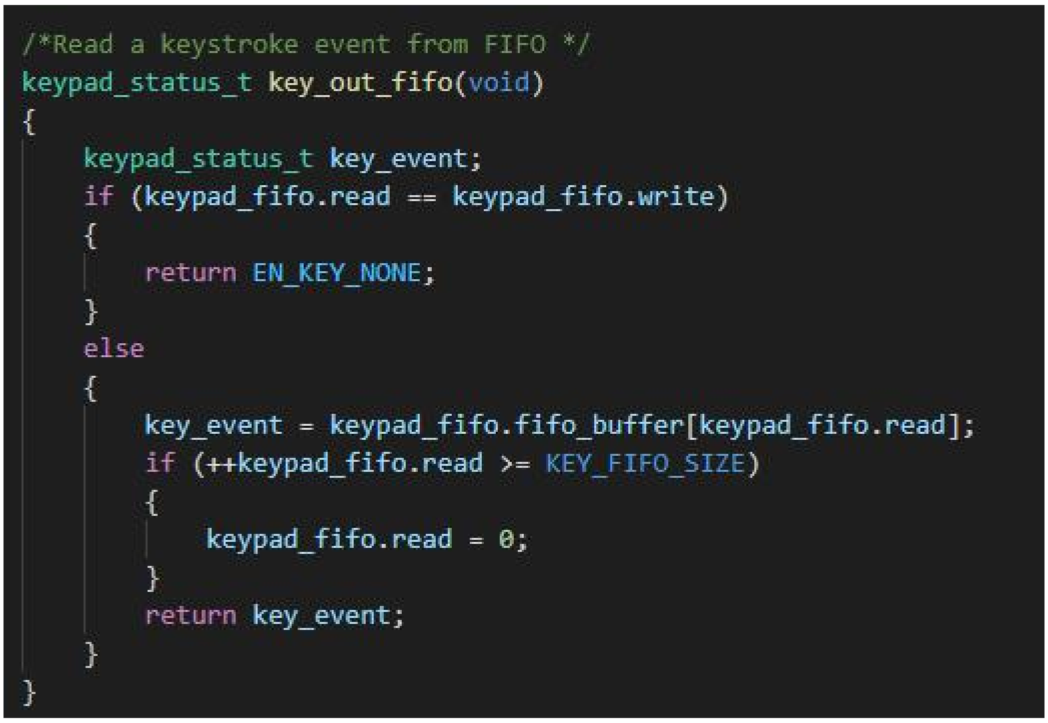
```
    p_key->prev_key_state = current_key_state;
}
```

4.8 Save the state of a key to the FIFO.

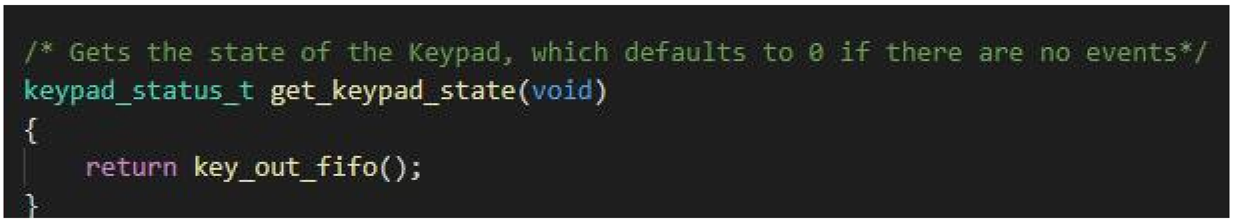


```
// Store the actual state of a key in FIFO
static void key_in_fifo(keypad_status_t keypad_status)
{
    keypad_fifo.fifo_buffer[keypad_fifo.write] = keypad_status;
    if (++keypad_fifo.write >= KEY_FIFO_SIZE)
    {
        keypad_fifo.write = 0;
    }
}
```

4.9 Read out an event of the keypad, the default is EN_KEY_NONE(0), other numbers correspond to different events.

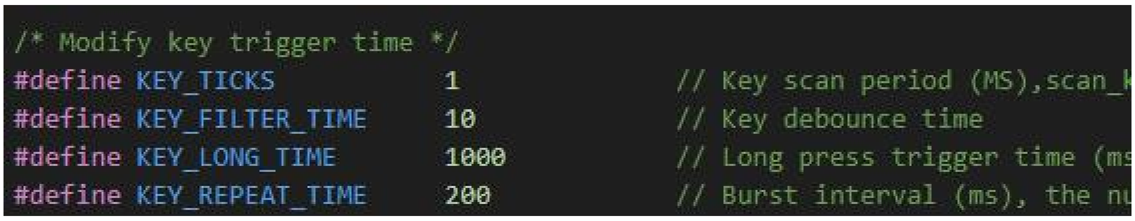


```
/*Read a keystroke event from FIFO */
keypad_status_t key_out_fifo(void)
{
    keypad_status_t key_event;
    if (keypad_fifo.read == keypad_fifo.write)
    {
        return EN_KEY_NONE;
    }
    else
    {
        key_event = keypad_fifo.fifo_buffer[keypad_fifo.read];
        if (++keypad_fifo.read >= KEY_FIFO_SIZE)
        {
            keypad_fifo.read = 0;
        }
        return key_event;
    }
}
```



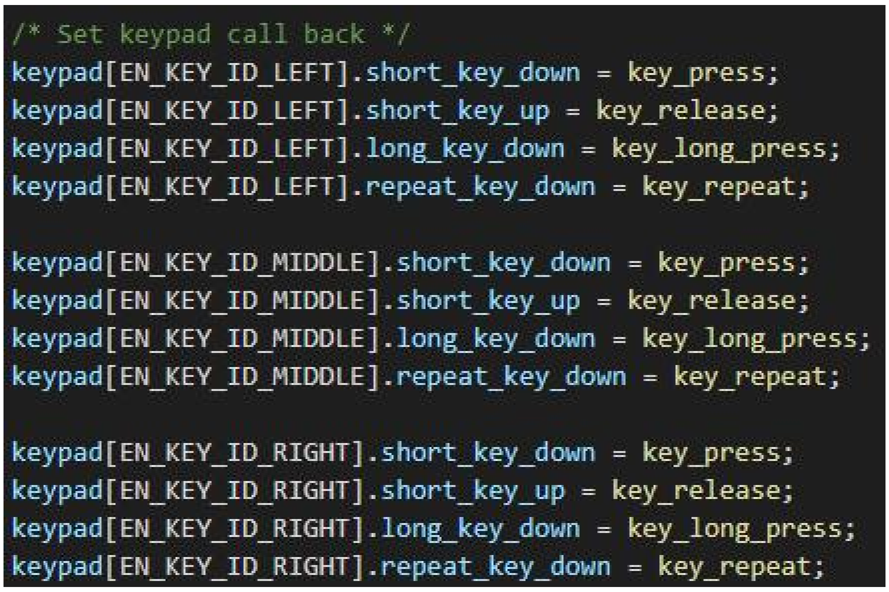
```
/* Gets the state of the Keypad, which defaults to 0 if there are no events*/
keypad_status_t get_keypad_state(void)
{
    return key_out_fifo();
}
```

4.10 If you need to modify the trigger time, you can modify the following parameters.



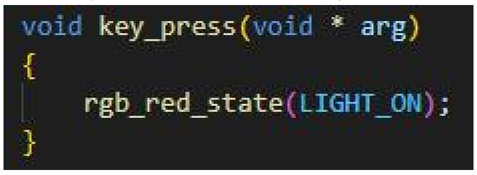
```
/* Modify key trigger time */
#define KEY_TICKS           1           // Key scan period (MS),scan_k
#define KEY_FILTER_TIME     10          // Key debounce time
#define KEY_LONG_TIME       1000        // Long press trigger time (ms
#define KEY_REPEAT_TIME     200         // Burst interval (ms), the nu
```

4.11 After initializing the keyboard, you can get the events of the keyboard. There are two ways in total. The first is by setting the callback function, and the second is by reading the state value.



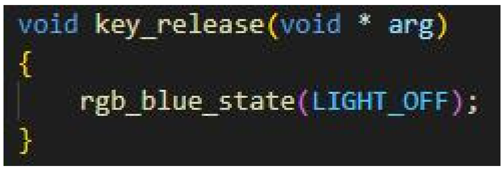
```
/* Set keypad call back */
keypad[EN_KEY_ID_LEFT].short_key_down = key_press;
keypad[EN_KEY_ID_LEFT].short_key_up = key_release;
keypad[EN_KEY_ID_LEFT].long_key_down = key_long_press;
keypad[EN_KEY_ID_LEFT].repeat_key_down = key_repeat;

keypad[EN_KEY_ID_MIDDLE].short_key_down = key_press;
keypad[EN_KEY_ID_MIDDLE].short_key_up = key_release;
keypad[EN_KEY_ID_MIDDLE].long_key_down = key_long_press;
keypad[EN_KEY_ID_MIDDLE].repeat_key_down = key_repeat;

keypad[EN_KEY_ID_RIGHT].short_key_down = key_press;
keypad[EN_KEY_ID_RIGHT].short_key_up = key_release;
keypad[EN_KEY_ID_RIGHT].long_key_down = key_long_press;
keypad[EN_KEY_ID_RIGHT].repeat_key_down = key_repeat;
```

When press button, red light is on.



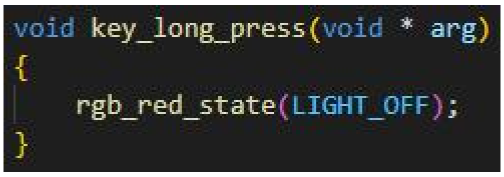
```
void key_press(void * arg)
{
    rgb_red_state(LIGHT_ON);
}
```

When press button, blue light is on.



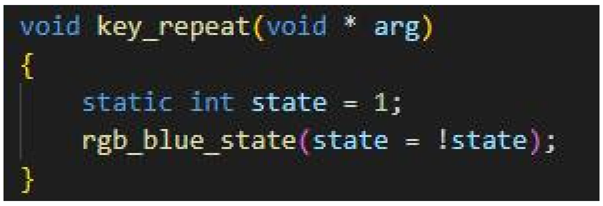
```
void key_release(void * arg)
{
    rgb_blue_state(LIGHT_OFF);
}
```

When long press button, red light is off.



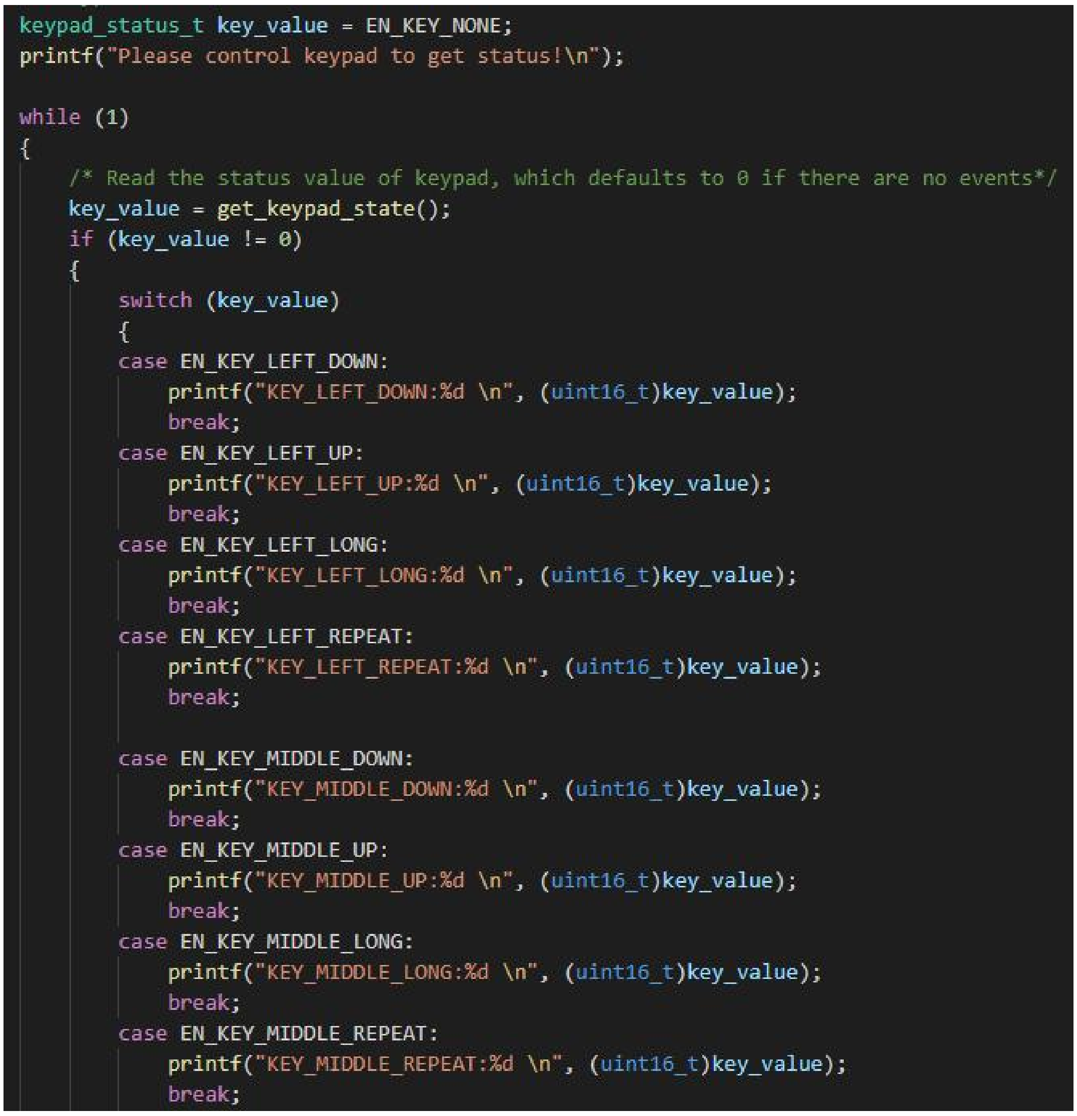
```
void key_long_press(void * arg)
{
    rgb_red_state(LIGHT_OFF);
}
```

Press the button continuously, blue light is flashing.

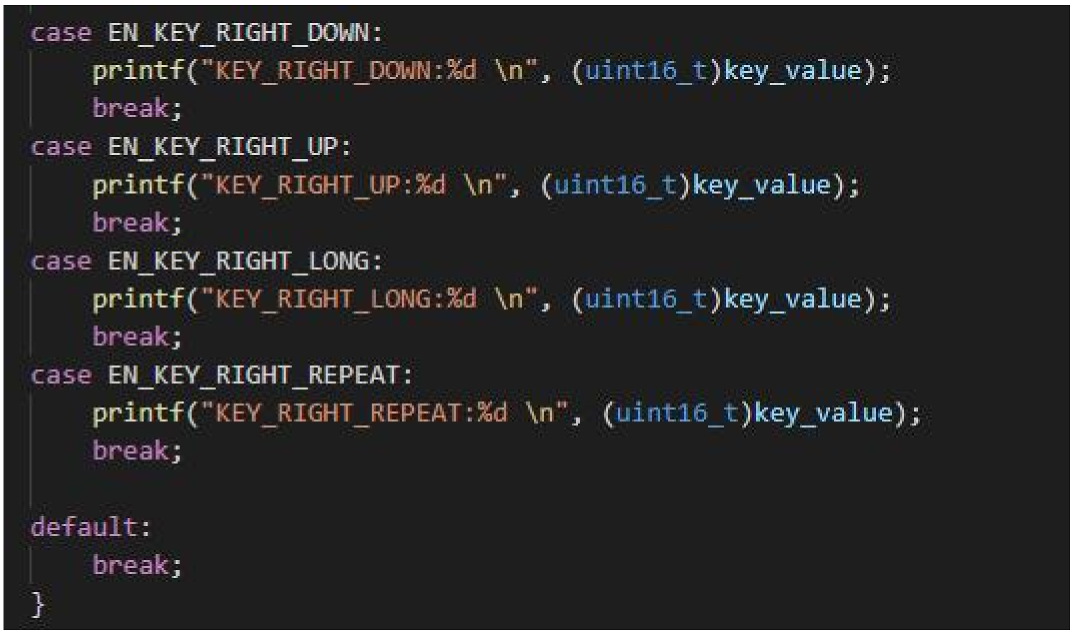


```
void key_repeat(void * arg)
{
    static int state = 1;
    rgb_blue_state(state = !state);
}
```

4.12 The second method: print the current button state value by reading the state value.



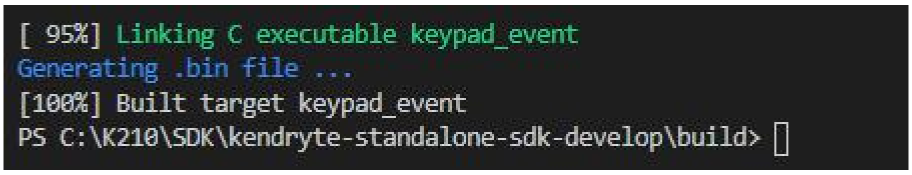
```c
keypad_status_t key_value = EN_KEY_NONE;
printf("Please control keypad to get status!\n");

while (1)
{
    /* Read the status value of keypad, which defaults to 0 if there are no events*/
    key_value = get_keypad_state();
    if (key_value != 0)
    {
        switch (key_value)
        {
        case EN_KEY_LEFT_DOWN:
            printf("KEY_LEFT_DOWN:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_LEFT_UP:
            printf("KEY_LEFT_UP:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_LEFT_LONG:
            printf("KEY_LEFT_LONG:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_LEFT_REPEAT:
            printf("KEY_LEFT_REPEAT:%d \n", (uint16_t)key_value);
            break;

        case EN_KEY_MIDDLE_DOWN:
            printf("KEY_MIDDLE_DOWN:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_MIDDLE_UP:
            printf("KEY_MIDDLE_UP:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_MIDDLE_LONG:
            printf("KEY_MIDDLE_LONG:%d \n", (uint16_t)key_value);
            break;
        case EN_KEY_MIDDLE_REPEAT:
            printf("KEY_MIDDLE_REPEAT:%d \n", (uint16_t)key_value);
            break;
```



```c
case EN_KEY_RIGHT_DOWN:
    printf("KEY_RIGHT_DOWN:%d \n", (uint16_t)key_value);
    break;
case EN_KEY_RIGHT_UP:
    printf("KEY_RIGHT_UP:%d \n", (uint16_t)key_value);
    break;
case EN_KEY_RIGHT_LONG:
    printf("KEY_RIGHT_LONG:%d \n", (uint16_t)key_value);
    break;
case EN_KEY_RIGHT_REPEAT:
    printf("KEY_RIGHT_REPEAT:%d \n", (uint16_t)key_value);
    break;

default:
    break;
}
```

13. Compile and debug, burn and run

Copy the keypad_event to the src directory in the SDK.
Then, enter the build directory and run the following command to compile.
cmake .. -DPROJ=keypad_event -G "MinGW Makefiles"
make

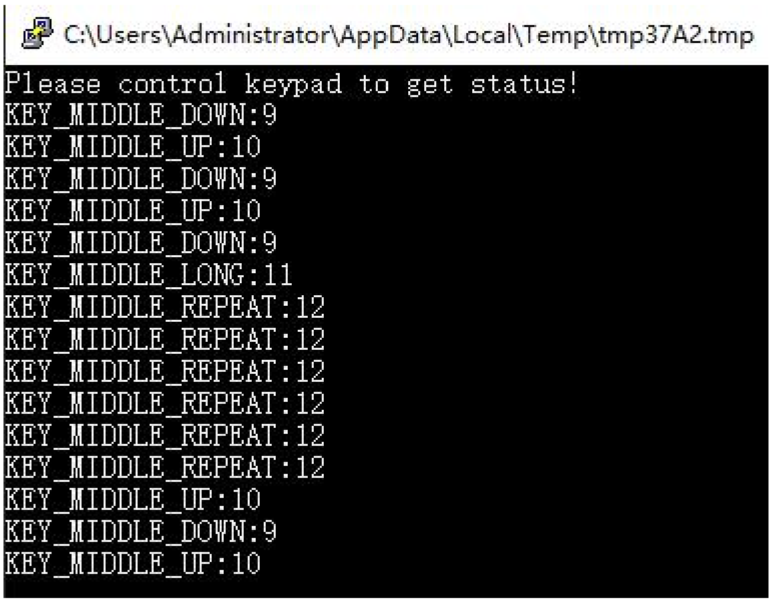


```
[ 95%] Linking C executable keypad_event
Generating .bin file ...
[100%] Built target keypad_event
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **keypad_event.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 **keypad_event.bin** file to the K210 development board.

5. Experimental phenomenon

After the firmware is write, 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.**
We can see that the terminal will print "Please control keypad to get status!"
When we press the middle button of keypad, the red light is on and the press prompt is printed. If it is released immediately, the release prompt will be printed, and the red light will not go out.
If you press and hold for about 1 second, the red light is off, and terminal interface will print the long-press prompt. If you continue to hold it down, it will enter the repeated printing event. The function is to print a repeated prompt every 0.2 seconds, the blue light flashes once every 0.4 seconds, and the blue light goes out when you release it.



```
C:\Users\Administrator\AppData\Local\Temp\tmp37A2.tmp
Please control keypad to get status!
KEY_MIDDLE_DOWN:9
KEY_MIDDLE_UP:10
KEY_MIDDLE_DOWN:9
KEY_MIDDLE_UP:10
KEY_MIDDLE_DOWN:9
KEY_MIDDLE_LONG:11
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_REPEAT:12
KEY_MIDDLE_UP:10
KEY_MIDDLE_DOWN:9
KEY_MIDDLE_UP:10
```

6. Experiment summary

6.1 There are three buttons inside the keypad, and only one button can be triggered at the same time.
6.2 By scanning the keyboard by the timer, the event of the keyboard can be detected and the callback function can be set.

6.3 The keyboard event can be obtained in two ways.
First method, set a callback function.
Second method, read the state value of the keyboard.