3.6 Timer experiment

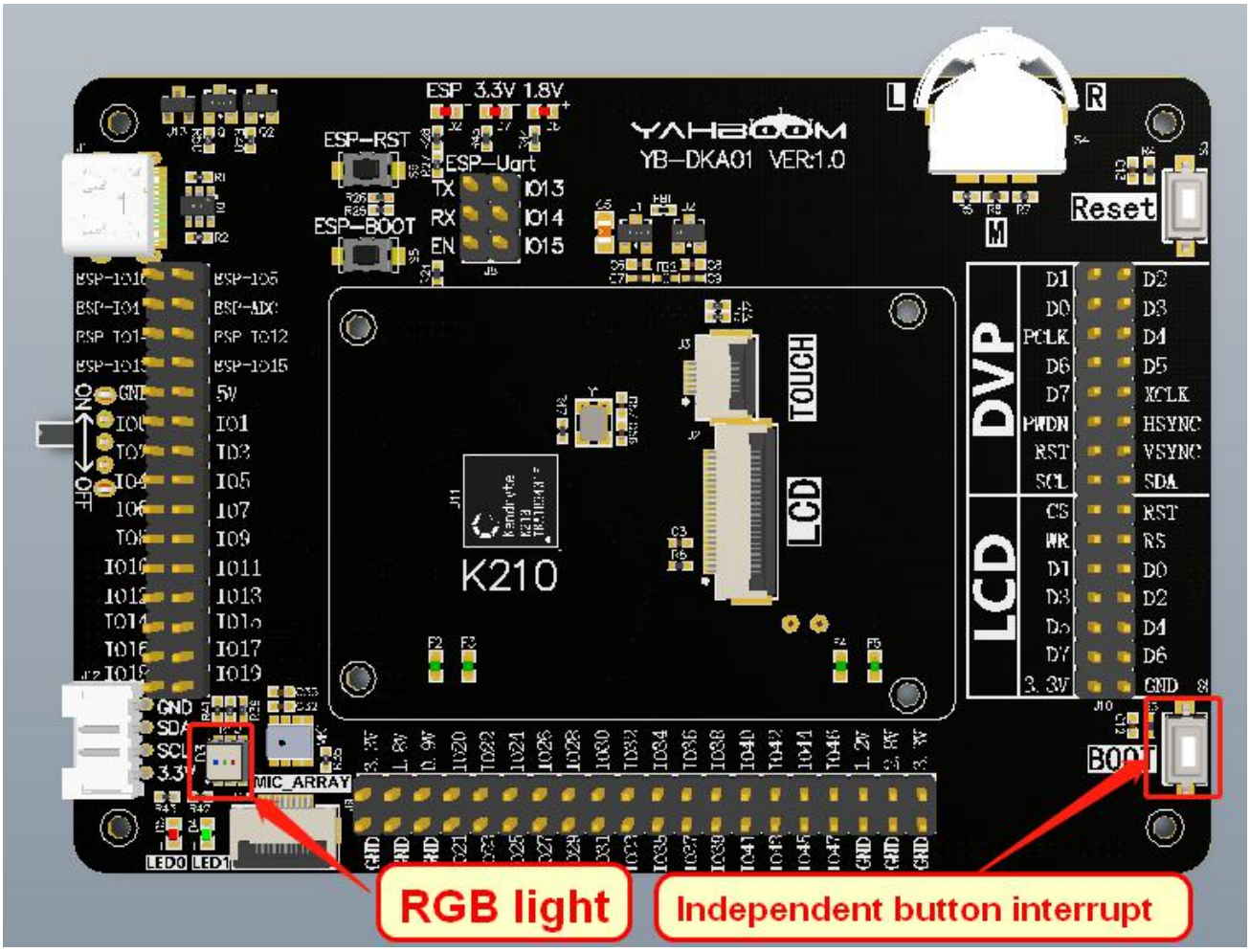
1. Experiment purpose

In this lesson, we mainly learns timer functions of K210.

2.Experiment preparation

2.1 components

Independent button BOOT, RGB light.

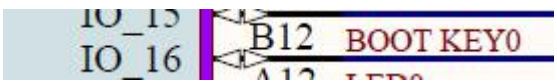


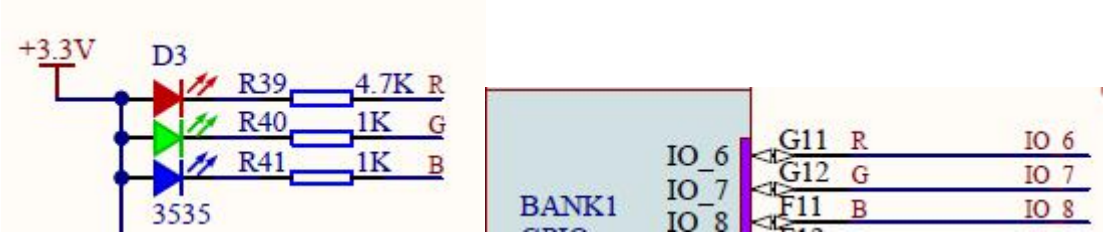
2.2 Component characteristics

There are a total of 3 chip timers, and each timer possess 4 channels. Each timer can set the trigger interval, and timer interrupt processing function.

2.3 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 timer.h

We will provide following interfaces for users:

- timer_init: Initialize the timer.
- timer_set_interval: Set the timing interval.
- timer_set_irq (no supported after 0.6.0 version, use timer_irq_register)
- timer_set_enable: Enable/disable timer.
- timer_irq_register: Register the timer interrupt callback function.
- timer_irq_deregister: Log off the timer interrupt.

3. Experimental principle

The core of the timer is actually to add 1 counter to count the machine cycles.

After each machine cycle, the counter automatically increments by 1 until the counter overflows.

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.

```
/*****************************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)

#endif /* _PIN_CONFIG_H_ */
```

```
void hardware_init(void)
{
    /* fpioa mapping */
    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.

```
/*External interrupt initialization*/
plic_init();
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 NULL.

```
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, NULL);

}
```

4.6 Each time BOOT button is pressed or released, the interrupt function key_irq_cb is triggered. In the interrupt, the current button state is read first, saved in key_state, and the timer state is set according to the state of key_state, and the timer is stopped when it is pressed. Turn on the timer when released.

```
int key_irq_cb(void* ctx)
{
    gpio_pin_value_t key_state = gpiohs_get_pin(KEY_GPIONUM);

    if (key_state)
        timer_set_enable(TIMER_NUM, TIMER_CHANNEL, 1);
    else
        timer_set_enable(TIMER_NUM, TIMER_CHANNEL, 0);
    return 0;
}
```

4.7 Initialize the timer, here is timer 0 channel 0, the timeout period is 500 milliseconds, the timer interrupt callback function is timer_timeout_cb, and the parameter is g_count.

```
void init_timer(void) {
    /* Initialize the timer */
    timer_init(TIMER_DEVICE_0);
    /* Set the timer timeout time, the unit is ns */
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 500 * 1e6);
    /* Set timer interrupt callback */
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_timeout_cb, &g_count);
    /* Enable timer */
    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

4.8 The processing in the timer interrupt, the on and off of the RGB light is modified each time it is interrupted. This function is equivalent to switching the RGB light on white or off every 0.5 seconds when the timer is on.

```
int timer_timeout_cb(void *ctx) {
    uint32_t *tmp = (uint32_t *)(ctx);
    (*tmp)++;
    if ((*tmp)%2)
    {
        rgb_all_on();
    }
    else
    {
        rgb_all_off();
    }
    return 0;
}
```

4.9 The function to turn on the RGB lamp to turn on white is rgb_all_on, that is, the three colors of the RGB lamp light up together to become white.

```c
void rgb_all_on(void)
{
    gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_LOW);
    gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_LOW);
    gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_LOW);
}
```

4.10 while loop.

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

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

    //Initialize RGB lights
    init_rgb();

    //Initialize the key
    init_key();

    //Initialize the timer
    init_timer();

    while (1);

    return 0;
}
```

4.11 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=timer -G "MinGW Makefiles"

make

```
[100%] Linking C executable timer
Generating .bin file ...
[100%] Built target timer
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

After the firmware is burned, a terminal interface will pop up. If the terminal interface does not pop up, you can open the serial port assistant to display the debugging content.
The RGB light turns on white, turns off after every 0.5 seconds, and then turns on again. It keeps cycling.
When the BOOT button is pressed, the timer stops. The RGB light saves the current state and no longer switches states.
When the BOOT button is released, the timer resumes. , RGB lights start to switch states every 0.5 seconds.

6. Experiment summary

6.1 The timer can set the timeout time of nanosecond level, and can set the interrupt callback.
6.2 The timer can be paused and restarted by controlling the enable and disable modes, without reconfiguration.
6.3 K210 has three timers in total, and each timer has four channels.

Appendix -- API

Header file is timer.h

timer_init

Description: Initialize the timer.
Function prototype: **void timer_init(timer_device_number_t timer_number)**
Parameter:

Parameter name	Description	Input/output
timer_number	Timer number	输入

Return value: No

timer_set_interval

Description: Set the timing interval.
Function prototype: **size_t timer_set_interval(timer_device_number_t timer_number, timer_channel_number_t channel, size_t nanoseconds)**
Parameter:

Parameter name	Description	Input/output

Parameter name	Description	Input/output
timer_number	Timer number	Input
channel	Timer channel number	Input
nanoseconds	Time interval (nanoseconds)	Input

Return value: The actual trigger interval (nanoseconds).

timer_set_irq

Description: Set the timer to trigger the interrupt callback function. This function is obsolete. The replacement function is timer_irq_register.

Function prototype: **void timer_set_irq(timer_device_number_t timer_number, timer_channel_number_t channel, void(*func)(), uint32_t priority)**

Parameter:

Parameter name	Description	Input/output
timer_number	Timer number	Input
channel	Timer channel number	Input
func	Callback function	Input
priority	Interrupt priority level	Input

Return value: No

timer_set_enable

Description: Enable and disable the timer.

Function prototype: **void timer_set_enable(timer_device_number_t timer_number, timer_channel_number_t channel, uint32_t enable)**

Parameter:

Parameter name	Description	Input/output
timer_number	Timer number	Input
channel	Timer channel number	Input
enable	Enable disable timer 0: disable 1: enable	Input

Return value: No

timer_irq_register

Description: The registered timer triggers the interrupt callback function.

Function prototype: **int timer_irq_register(timer_device_number_t device, timer_channel_number_t channel, int is_single_shot, uint32_t priority, timer_callback_t callback, void *ctx);**

Parameter

Parameter name	Description	Input/output
device	Timer number	Input

Parameter name	Description	Input/output
channel	Timer channel number	Input
is_single_shot	Whether it is a single interrupt	Input
priority	interrupt priority level	Input
callback	Interrupt callback function	Input
ctx	Call back function arguments	Input

Return value

Return value	Description
0	succeed
!0	fail

timer_irq_deregister

Description: Log off the timer interrupt function.

Function prototype: **int timer_irq_deregister(timer_device_number_t device, timer_channel_number_t channel)**

Parameter

Parameter name	Description	Input/output
device	Timer number	Input
channel	Timer channel number	Input

Return value

Return value	Description
0	succeed
!0	fail

Eg:

```
/*Timer 0 channel 0 timer 1 second print Time OK!*/
void irq_time(void)
{
    printf("Time OK!\n");
}
plic_init();
timer_init(TIMER_DEVICE_0);
timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1e9);
timer_set_irq(TIMER_DEVICE_0, TIMER_CHANNEL_0, irq_time, 1);
timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
sysctl_enable_irq();
```

Type of data

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

- timer_device_number_t: timer number
- timer_channel_number_t: timer channel number
- timer_callback_t: timer callback function

timer_device_number_t

Description: timer number

Define

```
typedef enum _timer_deivce_number
{
    TIMER_DEVICE_0,
    TIMER_DEVICE_1,
    TIMER_DEVICE_2,
    TIMER_DEVICE_MAX,
} timer_device_number_t;
```

Member

Member name	Description
TIMER_DEVICE_0	Timer 0
TIMER_DEVICE_1	Timer 1
TIMER_DEVICE_2	Timer 2

timer_channel_number_t

Description: channel number

Define

```
typedef enum _timer_channel_number
{
    TIMER_CHANNEL_0,
    TIMER_CHANNEL_1,
    TIMER_CHANNEL_2,
    TIMER_CHANNEL_3,
    TIMER_CHANNEL_MAX,
} timer_channel_number_t;
```

Member

Member name	Description
TIMER_CHANNEL_0	Timer channel 0
TIMER_CHANNEL_1	Timer channel 1
TIMER_CHANNEL_2	Timer channel 2
TIMER_CHANNEL_3	Timer channel 3

timer_callback_t

Description: Timer callback function.

Define

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