3.7 PWM breathing light

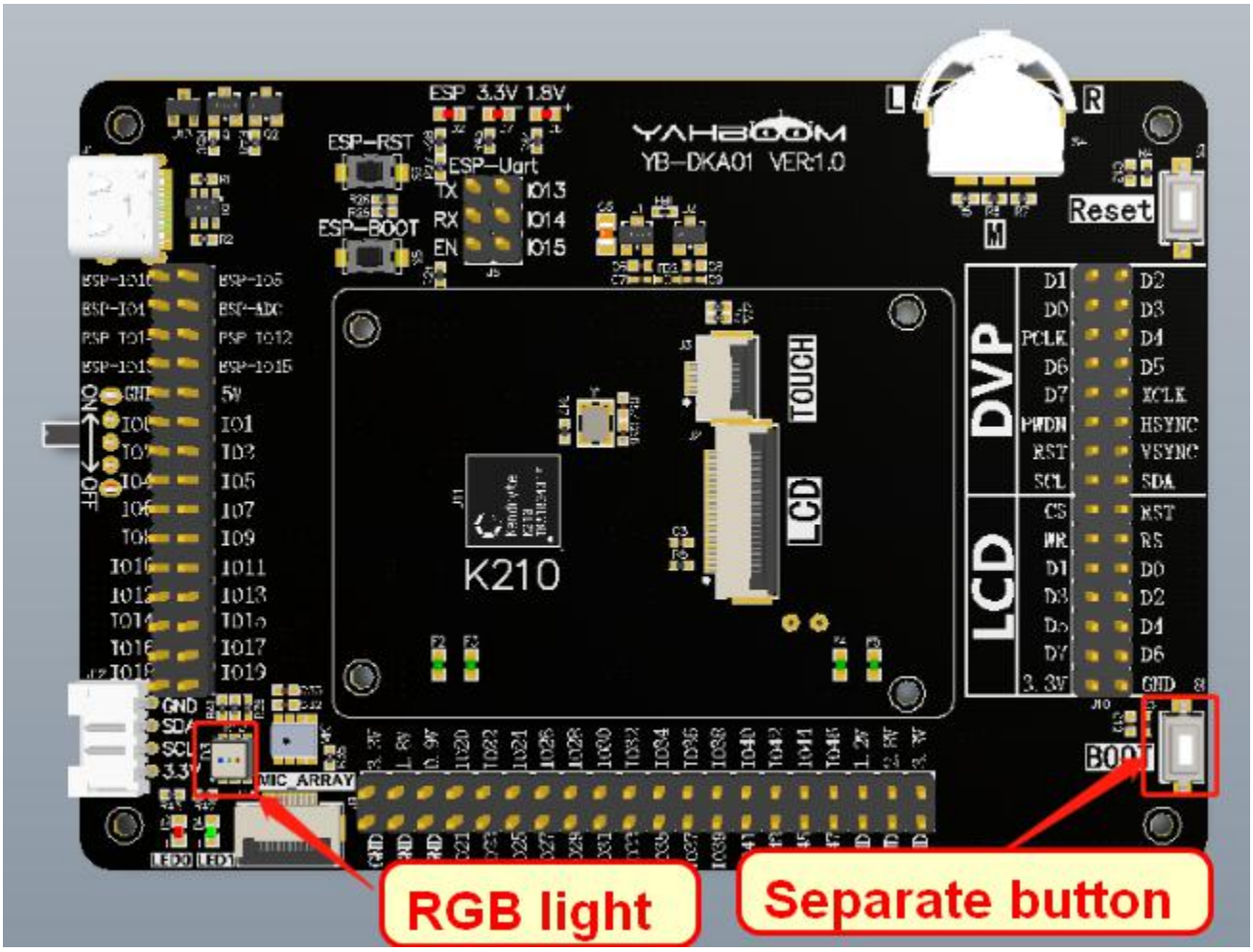
1. Experiment purpose

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

2.Experiment preparation

2.1 components

Separate button, BOOT LED light

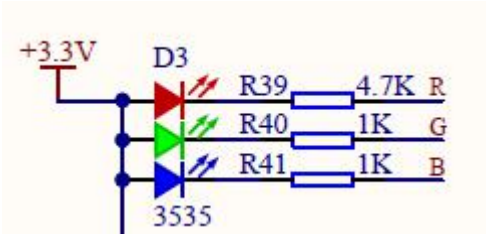


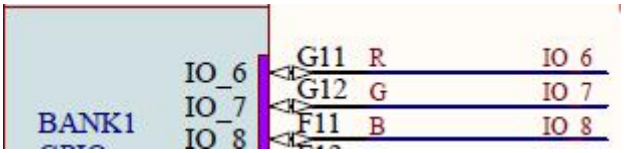
2.2 Component characteristics

There are 3 timers on the K210 chip, and each timer with 4 channels. Each timer can set the trigger interval, and timer interrupt processing function. The timer can also be set as the PWM output function, but if the PWM output is set, the timing function cannot be used.

2.3 Hardware connection

By default, the K210 development board has already welded the RGB lights. 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 pwm.h

PWM(pulse width modulator) is used to control the duty cycle of the pulse output. Its essence is a timer, so please pay attention to not conflict with the TIMER timer when setting the PWM number and channel.

We will provide following interfaces for users:

- pwm_init
- pwm_set_frequency
- pwm_set_enable

3. Experimental principle

PWM controls the duty cycle of the pulse output. The duty cycle refers to the ratio of the power-on time to the total time in a pulse cycle.

4. Experiment procedure

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

The mapping is channel 0 (switch 1) of timer 1 in here.

```
/*****************************HARDWARE-PIN******
// Hardware IO port, corresponding Schematic
#define PIN_RGB_R               (6)
#define PIN_RGB_G               (7)
#define PIN_RGB_B               (8)
```

```
void hardware_init(void)
{
    fpioa_set_function(PIN_RGB_R, FUNC_TIMER1_TOGGLE1);
}
```

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.

```
/* System interrupt initializ
plic_init();
sysctl_enable_irq();
```

4.3 Initialize the timer. The timer 0 channel 0 is used here, the timeout period is 10 milliseconds, the

timer interrupt callback function is timer_timeout_cb, the parameter is NULL.

```
void init_timer(void) {
    /*Init timer */
    timer_init(TIMER_DEVICE_0);
    /* Set the timer timeout time, unit:ns */
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 10 * 1e6);
    /* Set timer interrupt callback */
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_timeout_cb, NULL);
    /*enable timer */
    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

4.4 Initialize the PWM: timer 1 channel 0, a square wave with a frequency of 200KHz and a duty cycle of 0.5.

```
void init_timer(void) {
    /*Init timer */
    timer_init(TIMER_DEVICE_0);
    /* Set the timer timeout time, unit:ns */
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 10 * 1e6);
    /* Set timer interrupt callback */
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_timeout_cb, NULL);
    /*enable timer */
    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

4.5 Set the PWM duty cycle in the timer interrupt, and modify the PWM duty cycle according to the different values of duty_cycle to change the brightness of the RGB lamp.

```
int timer_timeout_cb(void *ctx) {
    static double duty_cycle = 0.01;
    /* 0为Gradually increase, 1为Gradually decrease */
    static int flag = 0;

    /*Pass in different values of cycle to adjust the occupancy ratio of
    pwm_set_frequency(PWM_DEVICE_1, PWM_CHANNEL_0, 200000, duty_cycle);

    /* Modify the value of cycle to increase and decrease gradually in t
    flag ? (duty_cycle -= 0.01): (duty_cycle += 0.01);
    if(duty_cycle > 1.0)
    {
        duty_cycle = 1.0;
        flag = 1;
    }
    else if (duty_cycle < 0.0)
    {
        duty_cycle = 0.0;
        flag = 0;
    }
    return 0;
}
```

4.6 while loop.

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

    /* System interrupt initialization and enable */
    plic_init();
    sysctl_enable_irq();

    /*Initialize the timer*/
    init_timer();

    /*Initialize PWM */
    init_pwm();

    while(1);

    return 0;
}
```

4.7 Compile and debug, burn and run

Copy the pwm to the src directory in the SDK.

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

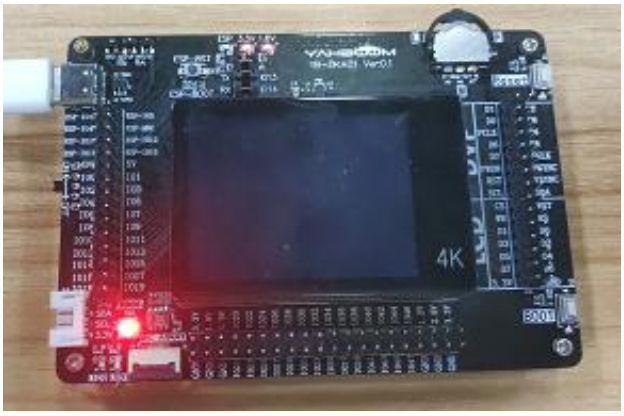
cmake .. -DPROJ=pwm -G "MinGW Makefiles"
make

```
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>
```

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. We can see that the RGB light is bright red, and it shows the effect of breathing light, the color gradually becomes darker from the brightest, and then gradually becomes brighter from the off state, and the cycle continues.




6. Experiment summary

6.1 The internal of PWM is based on the timer function.
6.2 Two important factors for controlling PWM are frequency and duty cycle.
6.3 The PWM output modifies the duty cycle, that is, the percentage of the power-on time to the total time. We can change the brightness of the RGB lamp by changing the effective value of the output.

Appendix -- API

Header file is pwm.h

pwm_init

Description: Initialize the PWM.
Function prototype: **void pwm_init(pwm_device_number_t pwm_number)**
Parameter:

Parameter name	Description	Input/Output
pwm_number	PWM number	Input

Return value: No

pwm_set_frequency

Description: set frequency and duty cycle.

Function prototype: **double pwm_set_frequency(pwm_device_number_t pwm_number, pwm_channel_number_t channel, double frequency, double duty)**

Parameter:

Parameter name	Description	Input/Output
pwm_number	PWM number	Input
channel	PWM channel number	Input
frequency	PWM output frequency	Input
duty	duty cycle	Input

Return value: Actual output frequency

pwm_set_enable

Description: enable/disable PWM.

Function prototype: **void pwm_set_enable(pwm_device_number_t pwm_number, uint32_t channel, int enable)**

Parameter:

Parameter name	Description	Input/Output
pwm_number	PWM number	Input
channel	PWM channel number	Input
enable	enable/disable PWM 0-disable 1-enable	Input

Return value: No

For example

```
/* pwm0 channel 1 outputs a 200KHZ square wave with a duty cycle of 0.5 */
/* Set IO13 as the output pin of PWM */
fpioa_set_function(13, FUNC_TIMER0_TOGGLE2);
pwm_init(PWM_DEVICE_0);
pwm_set_frequency(PWM_DEVICE_0, PWM_CHANNEL_1, 200000, 0.5);
pwm_set_enable(PWM_DEVICE_0, PWM_CHANNEL_1, 1);
```

type of data

- pwm_device_number_t: pwm number
- pwm_channel_number_t: pwm channel number

pwm_device_number_t

Description: pwm number.

Define

```
typedef enum _pwm_device_number
{
    PWM_DEVICE_0,
    PWM_DEVICE_1,
    PWM_DEVICE_2,
    PWM_DEVICE_MAX,
} pwm_device_number_t;
```

Member

Member name	Description
PWM_DEVICE_0	PWM0
PWM_DEVICE_1	PWM1
PWM_DEVICE_2	PWM2

pwm_channel_number_t

Description: pwm channel number

Define

```
typedef enum _pwm_channel_number
{
    PWM_CHANNEL_0,
    PWM_CHANNEL_1,
    PWM_CHANNEL_2,
    PWM_CHANNEL_3,
    PWM_CHANNEL_MAX,
} pwm_channel_number_t;
```

Member

Member name	Description
PWM_CHANNEL_0	PWM channel 0
PWM_CHANNEL_1	PWM channel 1
PWM_CHANNEL_2	PWM channel 2
PWM_CHANNEL_3	PWM channel 3