3.6 定时器实验

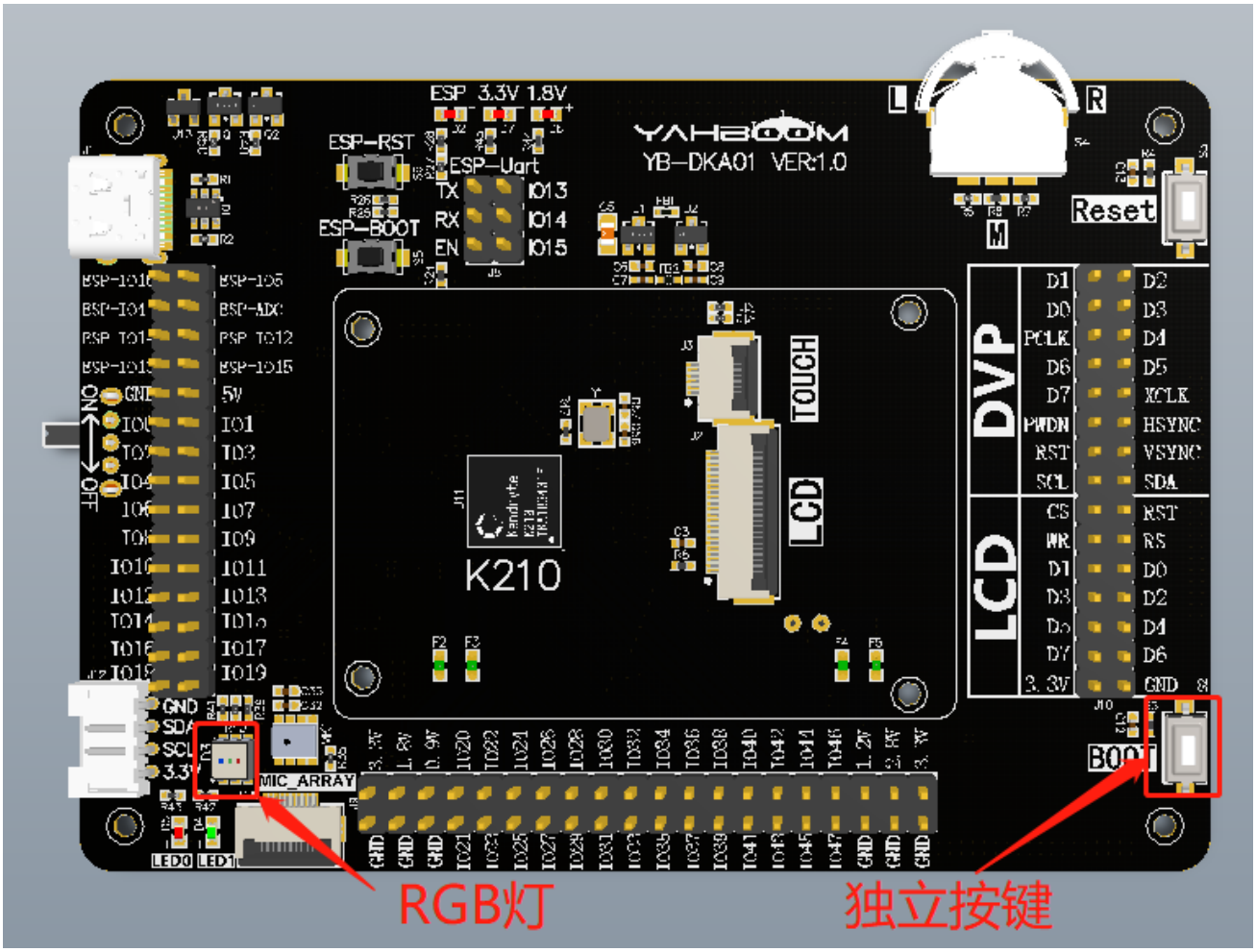
一、实验目的

本节课主要学习 K210 的定时器功能。

二、实验准备

1.实验元件

独立按键 BOOT、RGB 灯

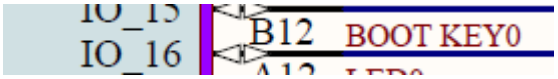


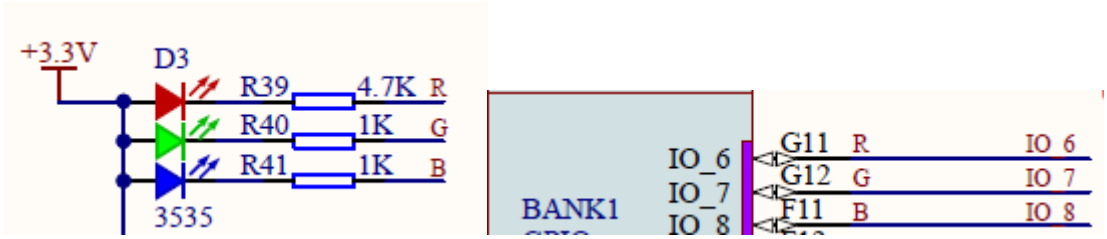
2.元件特性

K210 芯片定时器总共有 3 个，每个定时器有 4 路通道。每个定时器可以设置触发间隔，和定时器中断处理函数。

3.硬件连接

K210开发板出厂默认已经焊接好BOOT按键和RGB灯。按键连接的引脚为IO16。RGB灯R连接的是IO6，G连接的是IO7，B连接的是IO8。





4.SDK中对应API功能

对应的头文件 timer.h

为用户提供以下接口：

- timer_init：初始化定时器。
- timer_set_interval：设置定时间隔。
- timer_set_irq (0.6.0 后不再支持，请使用 timer_irq_register)
- timer_set_enable：使能/禁止定时器。
- timer_irq_register：注册定时器中断回调函数。
- timer_irq_deregister：注销定时器中断。

三、实验原理

定时器的核心其实是加1计数器，对机器周期进行计数，每过一个机器周期，计数器自动加1，直到计数器计满溢出。由于计数的周期是固定的，所以根据计数的多少就可以很方便的计算出计数的时间，当符合自己设定的超时时间，则调用中断回调函数，然后重新开始计数。

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。

这里要注意的是程序里操作的都是软件引脚，所以需要先把硬件引脚映射成软件 GPIO 功能，操作的时候直接操作软件 GPIO 即可。由于没有新增加硬件设备，所以初始化与上一节课是一样的。

```
/*****************************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_KEY                 (16)

/*****************************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 KEY_GPIONUM             (3)

/*****************************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_KEY                (FUNC_GPIOHS0 + KEY_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_KEY, FUNC_KEY);
}
```

2. 第二步需要初始化外部中断服务，并且使能全局中断。如果没有这一步操

作，系统的中断就不会运行，所以也不会调用中断回调函数。

```
/* 外部中断初始化 */
plic_init();
/* 使能全局中断 */
sysctl_enable_irq();
```

3. 在使用 RGB 灯前需要初始化，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();
}
```

4. 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);
}
```

5. 使用 BOOT 按键同样需要初始化，设置 BOOT 键为上拉输入模式，设置按键的 GPIO 电平触发模式为上升沿和下降沿，也可以设置单上升沿或单下降沿等，设置 BOOT 按键的中断回调函数为 key_irq_cb，参数为空 NULL。

```
void init_key(void)
{
    // 设置按键的GPIO模式为上拉输入
    gpiohs_set_drive_mode(KEY_GPIONUM, GPIO_DM_INPUT_PULL_UP);
    // 设置按键的GPIO电平触发模式为上升沿和下降沿
    gpiohs_set_pin_edge(KEY_GPIONUM, GPIO_PE_BOTH);
    // 设置按键GPIO口的中断回调
    gpiohs_irq_register(KEY_GPIONUM, 1, key_irq_cb, NULL);

}
```

6. 每次 BOOT 按下或者松开都会触发中断函数 key_irq_cb，在中断里先读取当前按键的状态，保存到 key_state 中，并且根据 key_state 的状态设置定时器的状态，当被按下时停止定时器，当被松开时打开定时器。

```
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;
}
```

7. 初始化定时器，这里使用的是定时器 0 通道 0，超时时间为 500 毫秒，定时器中断回调函数为 timer_timeout_cb，参数为 g_count。

```
void init_timer(void) {
    /* 定时器初始化 */
    timer_init(TIMER_DEVICE_0);
    /* 设置定时器超时时间，单位为ns */
    timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 500 * 1e6);
    /* 设置定时器中断回调 */
    timer_irq_register(TIMER_DEVICE_0, TIMER_CHANNEL_0, 0, 1, timer_timeout_cb, &g_count);
    /* 使能定时器 */
    timer_set_enable(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1);
}
```

8. 定时器中断内的处理，每次中断的时候修改 RGB 灯的亮灭。此函数相当于定时器打开的情况下，每 0.5 秒切换一次 RGB 灯点亮白色或熄灭的状态。

```
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;
}
```

9.点亮 RGB 灯亮白色的函数为 rgb_all_on，即 RGB 灯内部三种颜色的灯一起亮就变为白色。

```
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);
}
```

10.最后是一个 while(1)循环，这个是必须的，否则系统就会退出，不再运行。

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

    /* 初始化系统中断并使能 */
    plic_init();
    sysctl_enable_irq();

    /* 初始化RGB灯 */
    init_rgb();

    /* 初始化按键key */
    init_key();

    /* 初始化定时器 */
    init_timer();

    while (1);

    return 0;
}
```

11.编译调试，烧录运行

把本课程资料中的 timer 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=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>
```

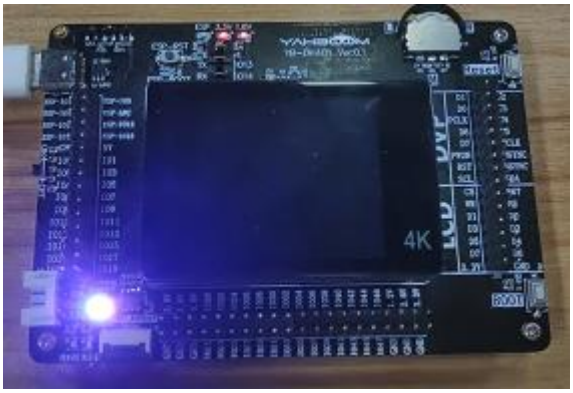
编译完成后，在 build 文件夹下会生成 timer.bin 文件。

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

烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以打开串口助手显示调试内容。

RGB 灯亮白色，每 0.5 秒后熄灭再亮白色，一直循环，当按住 BOOT 键时，定时器停止，RGB 灯保存当前的状态，不再切换状态，当松开 BOOT 键时，定时器恢复启动，RGB 灯又开始每 0.5 秒切换状态。



六、实验总结

1. 定时器可以设置纳秒级别的超时时间，并且可以设置中断回调。

2. 定时器可以通过控制使能与禁止的方式来暂停和重新启动，而不需要重新配置。

3. K210 总共有三个定时器，每个定时器有四个通道。

附：API

对应的头文件 timer.h

timer_init

描述

初始化定时器。

函数原型

```
void timer_init(timer_device_number_t timer_number)
```

参数

参数名称	描述	输入输出
timer_number	定时器号	输入

返回值

无。

timer_set_interval

描述

设置定时间隔。

函数原型

```
size_t timer_set_interval(timer_device_number_t timer_number,
timer_channel_number_t channel, size_t nanoseconds)
```

参数

参数名称	描述	输入输出
timer_number	定时器号	输入
channel	定时器通道号	输入
nanoseconds	时间间隔（纳秒）	输入

返回值

实际的触发间隔（纳秒）。

timer_set_irq

描述

设置定时器触发中断回调函数，该函数已废弃，替代函数为 timer_irq_register。

函数原型

```
void timer_set_irq(timer_device_number_t timer_number, timer_channel_number_t channel, void(*func)(), uint32_t priority)
```

参数

参数名称	描述	输入输出
timer_number	定时器号	输入
channel	定时器通道号	输入
func	回调函数	输入
priority	中断优先级	输入

返回值

无。

timer_set_enable

描述

使能禁用定时器。

函数原型

```
void timer_set_enable(timer_device_number_t timer_number, timer_channel_number_t channel, uint32_t enable)
```

参数

参数名称	描述	输入输出
timer_number	定时器号	输入
channel	定时器通道号	输入
enable	使能禁用定时器 0：禁用 1：使能	输入

返回值

无。

timer_irq_register

描述

注册定时器触发中断回调函数。

函数原型

```
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);
```

参数

参数名称	描述	输入输出
device	定时器号	输入
channel	定时器通道号	输入
is_single_shot	是否单次中断	输入
priority	中断优先级	输入
callback	中断回调函数	输入
ctx	回调函数参数	输入

返回值

返回值 描述
0 成功
非 0 失败

timer_irq_deregister

描述

注销定时器中断函数。

函数原型

```
int timer_irq_deregister(timer_device_number_t device, timer_channel_number_t channel)
```

参数

参数名称	描述	输入输出
device	定时器号	输入
channel	定时器通道号	输入

返回值

返回值	描述
0	成功
非 0	失败

举例

```
/* 定时器 0 通道 0 定时 1 秒打印 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();
```

数据类型

相关数据类型、数据结构定义如下：

- timer_device_number_t：定时器编号。
- timer_channel_number_t：定时器通道号。
- timer_callback_t：定时器回调函数。

timer_device_number_t

描述

定时器编号

定义

```
typedef enum _timer_deivce_number
{
```

```
    TIMER_DEVICE_0,
    TIMER_DEVICE_1,
    TIMER_DEVICE_2,
    TIMER_DEVICE_MAX,
} timer_device_number_t;
```

成员

成员名称	描述
TIMER_DEVICE_0	定时器 0
TIMER_DEVICE_1	定时器 1
TIMER_DEVICE_2	定时器 2

timer_channel_number_t

描述

定时器通道号。

定义

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

成员

成员名称	描述
TIMER_CHANNEL_0	定时器通道 0
TIMER_CHANNEL_1	定时器通道 1
TIMER_CHANNEL_2	定时器通道 2
TIMER_CHANNEL_3	定时器通道 3

timer_callback_t

描述

定时器回调函数。

定义

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