本次任务：

1、学习CH04示例程序，包括gpio.c和4个工程中的main.c.

2、给出 gpio\_set(LIGHT\_RED,LIGHT\_OFF);

语句中LIGHT\_RED和LIGHT\_OFF的值是多少?

贴出每一步的查找截图。

3、用直接地址编程方式，实现红绿蓝三灯轮流闪烁

4、用调用构件方式，实现红绿蓝的八种组合轮流闪烁

1. gpio.c分析：

这是GPIO底层驱动构件源文件，提供了一系列函数来初始化GPIO引脚、设置引脚状态、获取引脚状态、设置引脚的上拉/下拉电阻、设置引脚的驱动能力等功能。

gpio\_init(uint16\_t port\_pin, uint8\_t dir, uint8\_t state)：初始化指定端口引脚作为GPIO引脚功能，并定义为输入或输出，若是输出，还指定初始状态是低电平或高电平。

gpio\_set(uint16\_t port\_pin, uint8\_t state)：当指定端口引脚被定义为GPIO功能且为输出时，设定引脚状态。

gpio\_get(uint16\_t port\_pin)：当指定端口引脚被定义为GPIO功能且为输入时，获取指定引脚状态。

gpio\_reverse(uint16\_t port\_pin)：当指定端口引脚被定义为GPIO功能且为输出时，反转引脚状态。

gpio\_pull(uint16\_t port\_pin, uint8\_t pullselect)：当指定端口引脚被定义为GPIO功能且为输入时，设置引脚的上拉/下拉电阻。

gpio\_enable\_int(uint16\_t port\_pin,uint8\_t irqtype)：当指定端口引脚被定义为GPIO功能且为输入时，开启引脚中断，并设置中断触发条件。

gpio\_disable\_int(uint16\_t port\_pin)：当指定端口引脚被定义为GPIO功能且为输入时，关闭引脚中断。

gpio\_drive\_strength(uint16\_t port\_pin, uint8\_t control)：当引脚被配置为数字输出时，设置引脚的驱动能力。

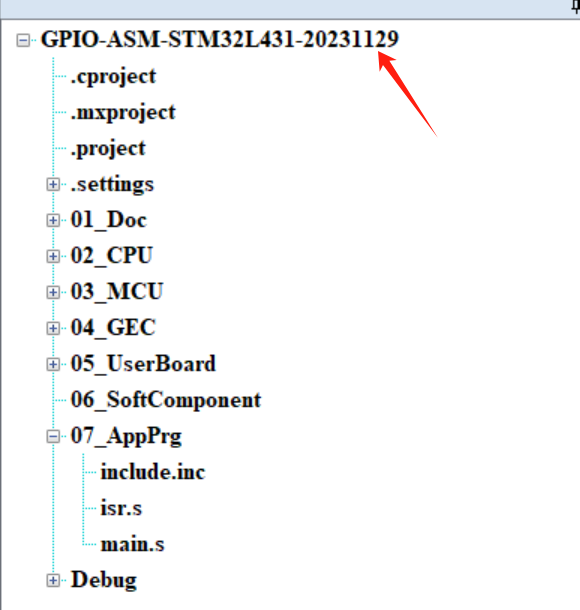
gpio\_get\_int(uint16\_t port\_pin)：当指定端口引脚被定义为GPIO功能且为输入时，获取中断标志。

gpio\_clear\_int(uint16\_t port\_pin)：当指定端口引脚被定义为GPIO功能且为输入时，清除中断标志。

gpio\_clear\_allint(void)：清除所有端口的GPIO中断。

gpio\_get\_port\_pin(uint16\_t port\_pin,uint8\_t\* port,uint8\_t\* pin)：解析端口号和引脚号。

main.s分析



1.

字符串定义：

在.data段定义了一些字符串，包括提示信息、数据格式控制符和灯的状态提示。

变量定义：

在.data段定义了一些变量，包括主循环次数变量 mMainLoopCount、灯的状态标志 mFlag 和灯的闪烁次数变量 mLightCount。

代码段定义：

在.text段定义了程序的执行逻辑。

主函数 main：

初始化工作，包括关闭总中断、初始化蓝灯和串口，启用串口中断，并打印欢迎提示信息。

设置主循环，循环执行主循环内的逻辑。

主循环：

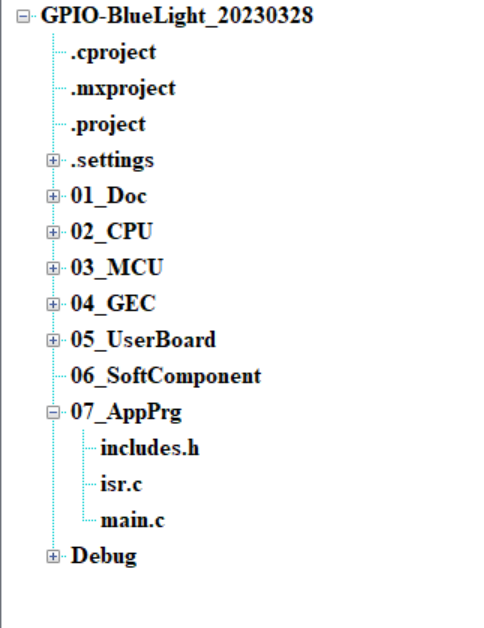
在主循环中，先累加主循环次数变量 mMainLoopCount，如果未达到预设的主循环次数，则继续循环。

当达到主循环次数后，根据灯的状态标志 mFlag，执行对应的操作：

如果灯的状态标志为 'L'，则增加灯的闪烁次数，并设置蓝灯亮，并打印相应提示信息。

如果灯的状态标志为 'A'，则设置蓝灯灭，并打印相应提示信息。

循环结束后，继续执行主循环。



包含头文件：引入了一个总头文件 includes.h，其中包含了所需的其他头文件。

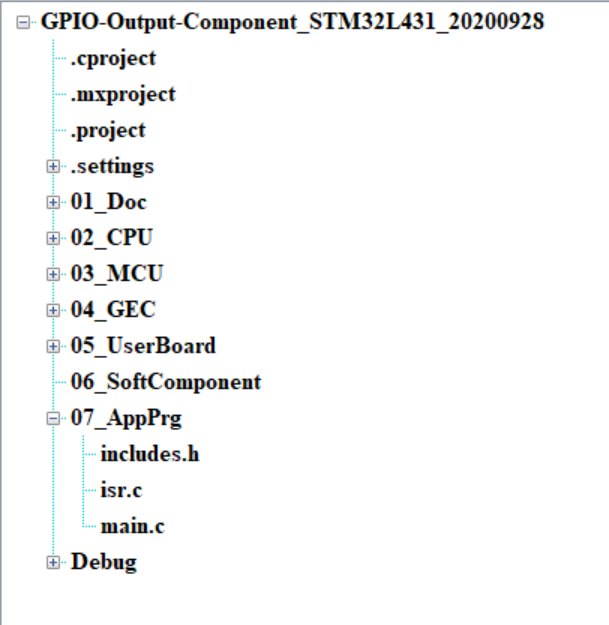
主函数 main()：

初始化红色LED，并关闭它。

打印一条提示信息，告诉用户程序已经开始运行，并且提示用户观察蓝色LED，此时应该是亮着的。

设置红色LED为亮。

进入一个无限循环，程序将一直停留在这里



主函数 main()：

启动部分（开头）：

声明并初始化了三个局部变量：mMainLoopCount、mFlag 和 mLightCount。

关闭总中断。

给局部变量赋初值。

初始化蓝色LED。

使能模块中断。

打印了一系列提示信息，向用户介绍程序的功能和使用方法。

主循环部分（开头）：

进入一个无限循环。

主循环次数变量 mMainLoopCount 自增。

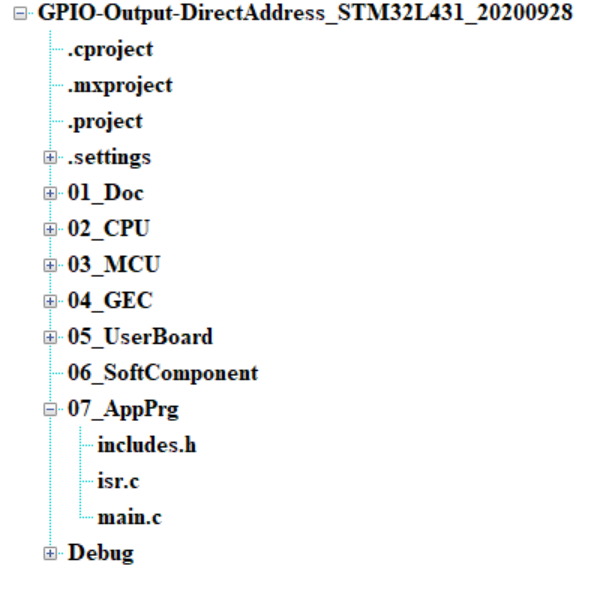
如果主循环次数未达到设定值，继续循环。

达到设定值时，进行灯的亮暗处理：

清除主循环次数变量。

如果灯的状态标志 mFlag 是 'L'，则将灯的闪烁次数加一并显示，然后将灯的状态标志改为 'A'，同时亮起蓝色LED，并打印灯亮的提示信息。

如果灯的状态标志 mFlag 不是 'L'，则将灯的状态标志改为 'L'，同时熄灭蓝色LED，并打印灯灭的提示信息。



主函数 main()：

启动部分（开头）：

声明并初始化了两个局部变量：mMainLoopCount（主循环次数）和 mFlag（蓝灯状态标志）。

关闭总中断。

给局部变量赋初值。

初始化蓝色LED。

使能模块中断。

打印了一系列提示信息，向用户介绍程序的功能和使用方法。

主循环部分（开头）：

进入一个无限循环。

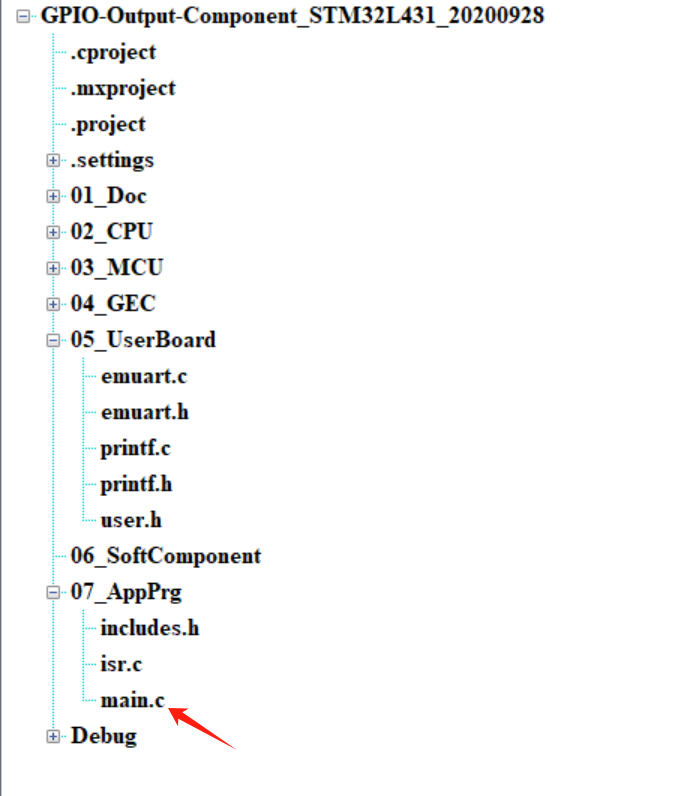
主循环次数自增，并判断是否小于特定常数。

若小于特定常数，则继续循环；否则，进行灯的亮暗切换：

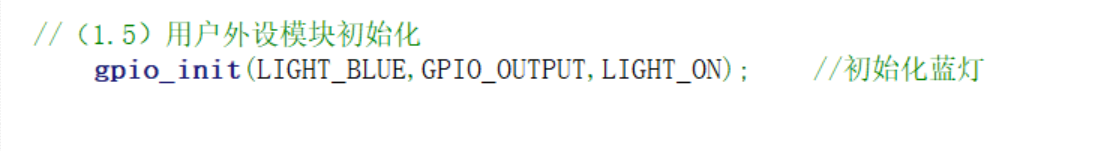
清除主循环次数。

若蓝灯状态标志 mFlag 为 'A'，则点亮蓝灯并打印信息，然后改变状态标志为 'L'；否则，熄灭蓝灯并打印信息，然后改变状态标志为 'A'。

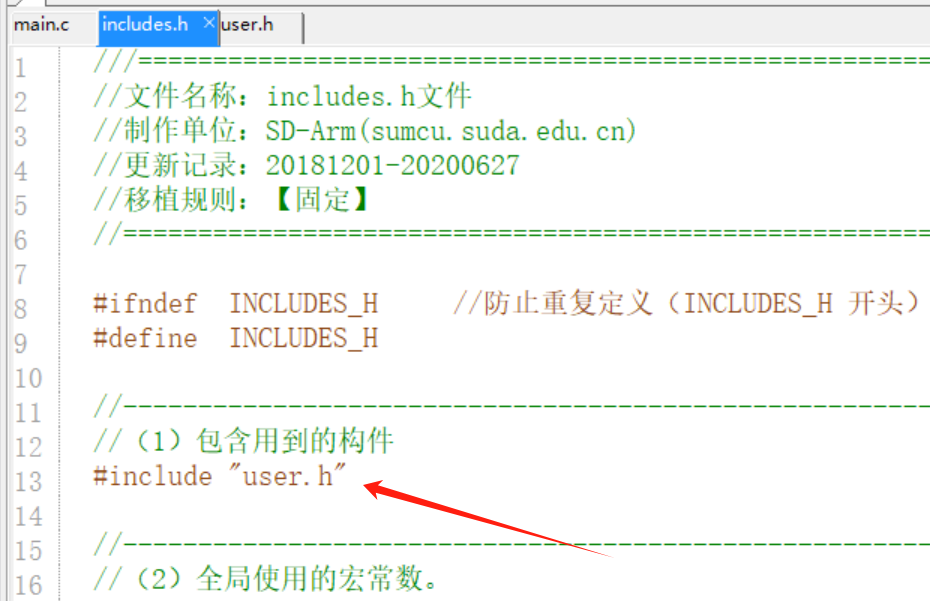
1. 先导入工程：



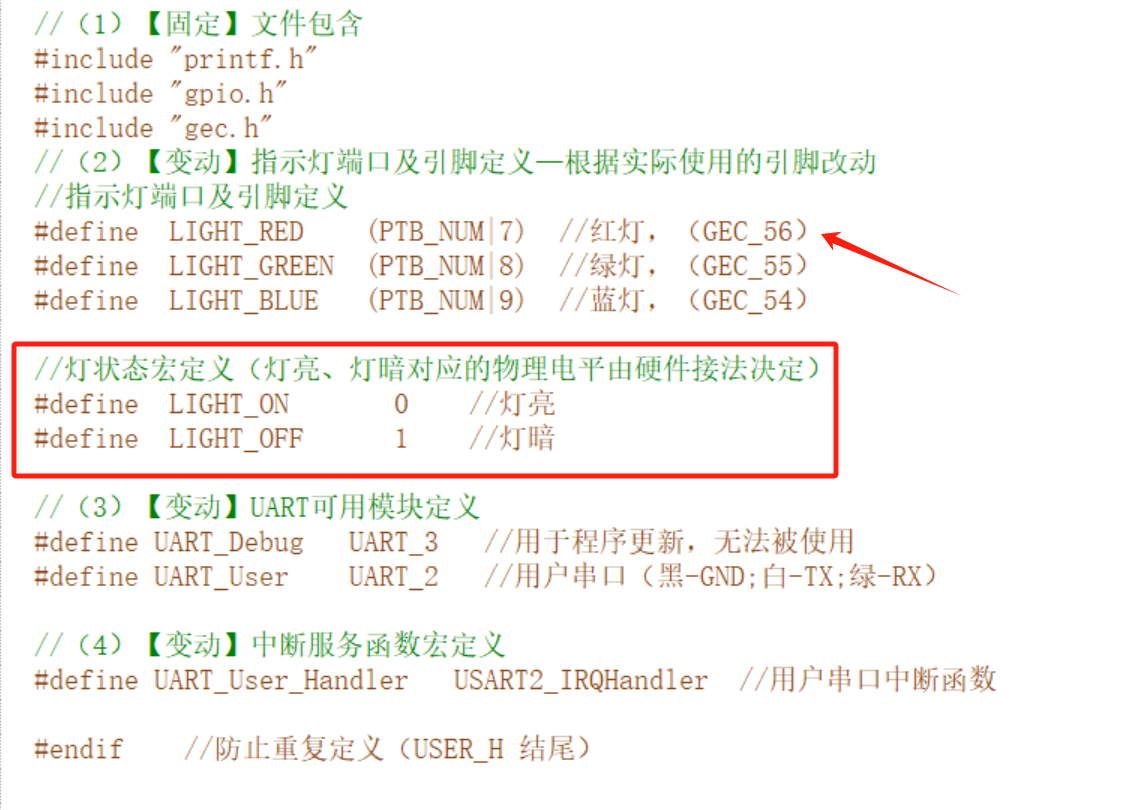
打开main会看到



根据代码中的注释和函数调用，可以猜测 LIGHT\_BLUE 和 LIGHT\_OFF 的值是通过宏定义得到的。因此，需要查看 includes.h 头文件以确定这些宏的定义。



再去user.h



这个PTB的定义在gpio.h



PTB\_NUM 的值是 1<<8，即左移 8 位，得到二进制 0000000100000000，转换为十进制为 256。

PTB\_NUM|7 即将 PTB\_NUM 的值与 7 进行按位或操作，得到二进制 0000000100000111，转换为十进制为 263。

1. 直接地址方式：

//====================================================================

//文件名称：main.c（应用工程主函数）

//框架提供：SD-Arm（sumcu.suda.edu.cn）

//版本更新：2017.08, 2020.05

//功能描述：见本工程的<01\_Doc>文件夹下Readme.txt文件

//====================================================================

#define GLOBLE\_VAR

#include "includes.h" //包含总头文件

//----------------------------------------------------------------------

//声明使用到的内部函数

//main.c使用的内部函数声明处

// 延时函数，单位为毫秒

void **delay**(uint32\_t milliseconds) {

**for** (uint32\_t i = 0; i < milliseconds \* 1000; ++i) {

// 空循环，用于消耗时间

**\_\_NOP**();

}

}

//----------------------------------------------------------------------

//主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程见书稿）

**int main**(void) {

// 初始化红、绿、蓝LED引脚为输出模式，并将它们的状态设置为灭

**volatile** uint32\_t\* RCC\_AHB2 = (uint32\_t\*)0x4002104C; // RCC\_AHB2 地址

**volatile** uint32\_t\* GPIOB\_MODER = (uint32\_t\*)0x48000400; // GPIOB\_MODER 地址

**volatile** uint32\_t\* GPIOB\_BSRR = (uint32\_t\*)0x48000418; // GPIOB\_BSRR 地址

// 使能相应 GPIOB 的时钟

\*RCC\_AHB2 |= (1 << 1); // GPIOB 的 B 口时钟使能

// 定义 B 口 7、8、9 脚为输出引脚

\*GPIOB\_MODER &= ~(0xFF << 14); // 清空 B 口 7、8、9 脚的 MODER 位

\*GPIOB\_MODER |= (0x55 << 14); // 设置 B 口 7、8、9 脚为输出模式

// 循环控制 LED 闪烁

**while**(1){

// 红灯亮，其余灭

\*GPIOB\_BSRR = (1 << 23); // 复位 B 口 8、9 脚（灭）

**delay**(6000);

\*GPIOB\_BSRR = (1 << 7)|(1 << 8)|(1 << 9); // 设置 B 口 7 脚为高电平（亮）

**delay**(6000);

//绿灯亮，其余灭

\*GPIOB\_BSRR = (1 << 24); // 复位 B 口 7、9 脚（灭）

**delay**(6000);

\*GPIOB\_BSRR = (1 << 7)|(1 << 8)|(1 << 9); // 设置 B 口 8 脚为高电平（亮）

**delay**(6000);

\*GPIOB\_BSRR = (1 << 25); // 复位 B 口 7、8 脚（灭）

**delay**(6000);

\*GPIOB\_BSRR = (1 << 7)|(1 << 8)|(1 << 9); // 设置 B 口 9 脚为高电平（亮）

**delay**(6000);

}

return 0;

}

1. 构件方式：

RGB跑马灯：(这个很好玩)

//======================================================================

//文件名称：main.c（应用工程主函数）

//框架提供：苏州大学嵌入式系统与物联网研究所（sumcu.suda.edu.cn）

//版本更新：20191108-20230328

//功能描述：见本工程的..\01\_Doc\Readme.txt

//移植规则：【固定】

//======================================================================

#define GLOBLE\_VAR

#include "includes.h" //包含总头文件

//----------------------------------------------------------------------

//声明使用到的内部函数

//main.c使用的内部函数声明处

// 延时函数，单位为毫秒

void **delay**(uint32\_t milliseconds) {

**for** (uint32\_t i = 0; i < milliseconds \* 1000; ++i) {

// 空循环，用于消耗时间

**\_\_NOP**();

}

}

//----------------------------------------------------------------------

//主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程）

**int main**(void) {

// 初始化红、绿、蓝LED引脚为输出模式，并将它们的状态设置为灭

**gpio\_init**(LIGHT\_RED, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_GREEN, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_BLUE, GPIO\_OUTPUT, LIGHT\_OFF);

// 循环控制LED的闪烁

**for** (;;) {

// 红灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

// 绿灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

// 蓝灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

// 延时3秒

**delay**(3000);

}

return 0;

}

这个是组合颜色：

//======================================================================

//文件名称：main.c（应用工程主函数）

//CS212 LHL

//======================================================================

#define GLOBLE\_VAR

#include "includes.h" //包含总头文件

//----------------------------------------------------------------------

//声明使用到的内部函数

//main.c使用的内部函数声明处

// 延时函数，单位为毫秒

void **delay**(uint32\_t milliseconds) {

**for** (uint32\_t i = 0; i < milliseconds \* 1000; ++i) {

// 空循环，用于消耗时间

**\_\_NOP**();

}

}

//----------------------------------------------------------------------

//主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程）

**int main**(void) {

// 初始化红、绿、蓝LED引脚为输出模式，并将它们的状态设置为灭

**gpio\_init**(LIGHT\_RED, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_GREEN, GPIO\_OUTPUT, LIGHT\_OFF);

**gpio\_init**(LIGHT\_BLUE, GPIO\_OUTPUT, LIGHT\_OFF);

// 循环控制LED的闪烁

**for** (;;) {

// 红灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

// 绿灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

// 蓝灯亮，其余灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

// 延时3秒

**delay**(3000);

// 红、绿同时亮，蓝灭

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

// 红、蓝同时亮，绿灭

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

// 延时3秒

**delay**(3000);

// 绿、蓝同时亮，红灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

// 延时3秒

**delay**(3000);

// 红、绿、蓝同时亮

**gpio\_set**(LIGHT\_RED, LIGHT\_ON);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_ON);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_ON);

// 延时3秒

**delay**(3000);

// 全部熄灭

**gpio\_set**(LIGHT\_RED, LIGHT\_OFF);

**gpio\_set**(LIGHT\_GREEN, LIGHT\_OFF);

**gpio\_set**(LIGHT\_BLUE, LIGHT\_OFF);

// 延时3秒

**delay**(3000);

}

return 0;

}