第4章 汇编语言框架（二）

4.3 认识工程框架中的GPIO构件

4.4 实验一：理解汇编程序框架及运行

4.3 认识工程框架中的GPIO构件

4.3.1 通常I/O接口基本概念及连接方法

1. I/O接口的概念

I/O接口，即输入/输出（Input/Output）接口，是MCU同外界进行交互的重要通道，MCU与外部设备的数据交换通过I/O接口来实现。I/O接口是一电子电路，其内有若干专用寄存器和相应的控制逻辑电路构成。

2. 通用I/O（GPIO）

通用I/O，也记为GPIO（General Purpose I/O），即基本的输入/输出，有时也称并行I/O，或普通I/O，它是I/O的最基本形式。本书中使用正逻辑，电源（Vcc）代表高电平，对应数字信号“1”；地（GND）代表低电平，对应数字信号“0”。

补充：高低电平的实际物理值

TTL电平：transistor transistor logic（晶体管-晶体管逻辑电平）。数字电路中，由TTL电子元器件组成电路使用的电平。电平是个电压范围，规定输出高电平>2.4V,输出低电平<0.4V。在室温下，一般输出高电平是3.5V，输出低电平是0.2V。最小输入高电平和低电平：输入高电平>=2.0V，输入低电平<=0.8V，噪声容限是0.4V。

目前：3.3V供电、1.8供电，电平实际物理值相对改变

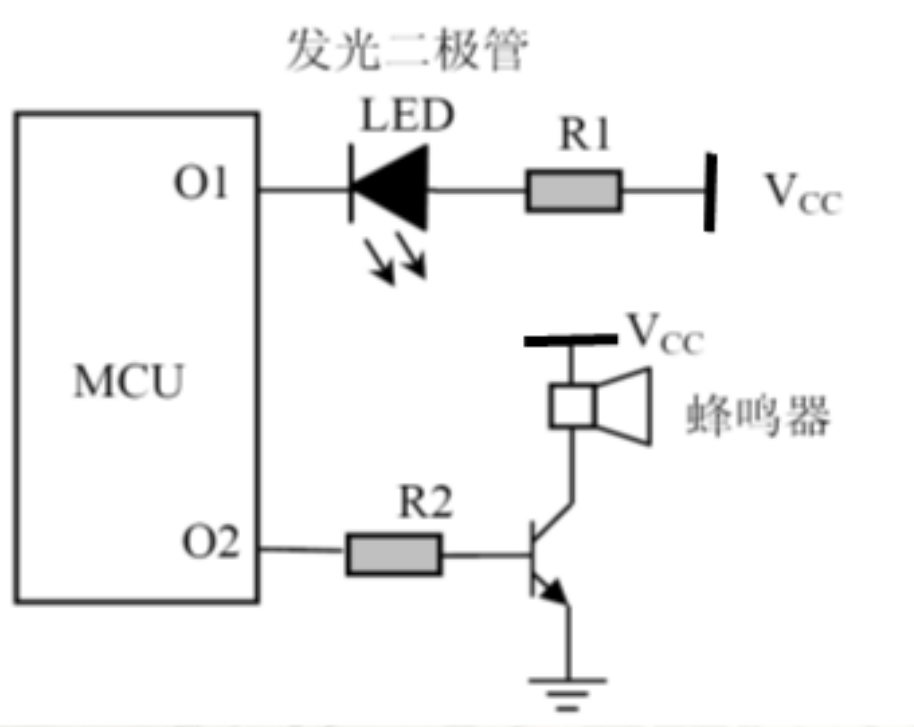
课后札记查阅：CMOS电平、232电平、485电平

3. 输出引脚的基本接法

作为通用输出引脚，MCU内部程序向该引脚输出高电平或低电平来驱动器件工作，即开关量输出。输出引脚O1和O2采用了不同的方式驱动外部器件。

一种接法是O1直接驱动发光二极管LED，当O1引脚输出高电平时，LED不亮；当O1引脚输出低电平时，LED点亮。这种接法的驱动电流一般在2mA～10mA。

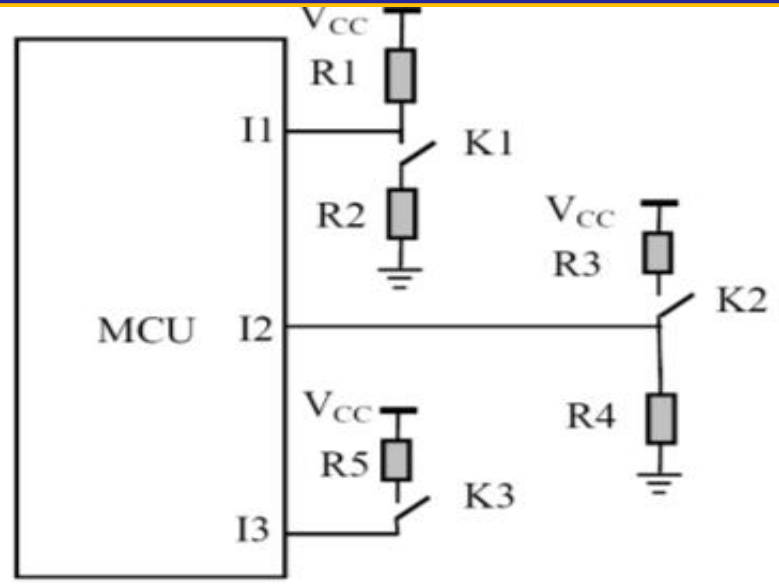
另一种接法是O2通过一个NPN三极管驱动蜂鸣器，当O2引脚输出高电平时，三极管导通，蜂鸣器响；当O2引脚输出低电平时，三极管截止，蜂鸣器不响。这种接法可以用O2引脚上的几个mA的控制电流驱动高达100mA的驱动电流。若负载需要更大的驱动电流，就必须采用光电隔离外加其他驱动电路，



提示：对于GPIO输出，要有负载（功率）的概念

4. 上拉下拉电阻与输入引脚的基本接法

芯片输入引脚的外部有三种不同的连接方式：带上拉电阻的连接、带下拉电阻的连接和“悬空”连接。通俗地说，若MCU的某个引脚通过一个电阻接到电源（Vcc）上，这个电阻被称为“上拉电阻”；与之相对应，若MCU的某个引脚通过一个电阻接到地（GND）上，则相应的电阻被称为“下拉电阻”。这种做法使得，悬空的芯片引脚被上拉电阻或下拉电阻初始化为高电平或低电平。



引脚I1通过上拉电阻R1接到Vcc，选择R1>>R2，K1断开时，引脚I1为高电平，K1闭合时，引脚I1为低电平。

引脚I2通过下拉电阻R4接到地，选择R3<<R4，K2断开时，引脚I2为低电平，K2闭合时，引脚I2为高电平。

引脚I3处于悬空状态，K3断开时，引脚I3的电平不确定（这样不好）。

提示：对于GPIO输入，要有引脚吸纳多少电流的概念

4.3.2 GPIO构件知识要素分析

GPIO初始化函数的参数应该有哪些？

由于芯片引脚具有复用特性，对引脚必须做如下操作：

1、使能GPIO时钟

2、应把引脚设置成GPIO功能

3、同时定义成输入或输出；若是输出，还要给出初始状态。

所以GPIO模块初始化函数gpio_init的参数有3个，分别为引脚、引脚方向和引脚状态。

其他的GPIO函数也可以采用类似的方法进行分析。

表4-4 GPIO驱动构件要素

序号	函数			形参		宏 常数	备注
	简明功能	返回	函数名	英文名	中文名		
1	初始化	无	gpio_init	port_pin	引脚端口号	用	采用端口号(引脚号)
				dir	引脚方向	用	
				state	引脚状态	用	
2	设置引脚状态	无	gpio_set	port_pin	引脚端口号	用	
				state	引脚状态	用	
3	获得引脚状态	引脚状态： 1=高电平 0=低电平	gpio_get	port_pin	引脚端口号	用	
4	引脚状态反转	无	gpio_reverse	port_pin	引脚端口号	用	
5	引脚上下拉使能	无	gpio_pull	port_pin	引脚端口号	用	
				pullselect	上拉/下拉	用	
6	使能引脚中断	无	gpio_enable_int	port_pin	引脚端口号	用	
				irqtype	引脚中断类型	用	
7	禁用引脚中断	无	gpio_disable_int	port_pin	引脚端口号	用	

补充：STM32L431-实验板芯片的GPIO相关知识

1、GPIO port mode register (GPIOx_MODER) (x =A to E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)

Reset value:

- 0xABFF FFFF (for port A)
- 0xFFFF FEBF (for port B)
- 0xFFFF FFFF for ports C..E
- 0x0000 000F (for port H)

补充：STM32L431-实验板芯片的GPIO相关知识

1、AHB2 peripheral clock enable register (RCC_AHB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	res.	AESEN (1)
													rw		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	res.	ADCEN	res.	Res.	Res.	Res.	res.	GPIOH EN	res.	res.	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw						rw			rw	rw	rw	rw	rw

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

Bit 0 **GPIOAEN:** IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

数据手册P68：关于地址映射

	0x4002 1000 - 0x4002 13FF	1 KB	RCC	*Section 6.4.32: RCC register map*

Address offset: 0x4C

Reset value: 0x0000 0000

补充：STM32L431-实验板芯片的GPIO相关知识

2、GPIO port input data register (GPIOx_IDR) (x = A to E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ID[15:0]:** Port x input data I/O pin y (y = 15 to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

Address offset: 0x10

Reset value: 0x0000 XXXX

补充：STM32L431-实验板芯片的GPIO相关知识

3、GPIO port output data register (GPIOx_ODR) (x = A to E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OD[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the OD bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers (x = A..F).

Address offset: 0x14

Reset value: 0x0000 0000

补充：STM32L431-实验板芯片的GPIO相关知识

4、GPIO port bit set/reset register (GPIOx_BSRR) (x = A to E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BR[15:0]:** Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODx bit

1: Resets the corresponding ODx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BS[15:0]:** Port x set I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODx bit

1: Sets the corresponding ODx bit

Address offset: 0x18

Reset value: 0x0000 0000

补充：STM32L431-实验板芯片的GPIO相关知识

5、GPIO port bit reset register (GPIOx_BRR) (x = A to E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **BR[15:0]:** Port x reset IO pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODx bit

1: Reset the corresponding ODx bit

Address offset: 0x28

Reset value: 0x0000 0000

补充：GPIO构件函数解析

1、gpio_port_pin_resolution:

```
//=============================================================================
//函数名称：gpio_port_pin_resolution
//函数返回：无
//参数说明：r0：端口号|引脚号,例:(PTB_NUM|(5u))表示B口5脚,头文件中有宏定义
//功能概要：将传进参数r0进行解析，得出具体端口号与引脚号（如：PORTB|(5)
//          解析为PORTB与5，并将其分别赋值给r0与r1）。
//=============================================================================
```

补充：GPIO构件函数解析

1、gpio_port_pin_resolution:

```
gpio_port_pin_resolution:
//（1）保存现场
        push {lr}                   //lr进栈（lr中为进入中断前pc的值）
//（2）解析入口参数r0：端口号|引脚号，得到具体端口号和引脚号，并将其分别赋值给r0与r1
        mov r4,r0                   //r4=r0=端口号|引脚号
        mov r5,r0                   //r5=r0=端口号|引脚号
        lsr r4,#8                   //逻辑左移获得端口号，r4=端口号
        mov r0,r4                   //r0=r4=端口号    //yws@2021-04-13将返回值存入r0
        mov r6,#0x000000ff
        and r5,r6                   //r5=引脚号
        mov r1,r5                   //r1=r5=引脚号    //yws@2021-04-13将返回值存入r1
//（3）恢复现场
        pop {pc}                    //恢复现场，lr出栈到pc（即子程序返回）
```

补充：GPIO构件函数解析

2、gpio_init:

```
//=========================================================================
// 函数名称：gpio_init
// 函数返回：无
// 参数说明：r0:(端口号|(引脚号)),例:(PTB_NUM|(5u))表示B口5脚,头文件中有宏定义
//          r1:引脚方向（0=输入,1=输出,可用引脚方向宏定义）
//          r2:端口引脚初始状态（0=低电平，1=高电平）
// 功能概要：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
// 备    注：端口x的每个引脚控制寄存器PORTx_PCRn的地址=PORT_PCR_BASE+x*0x1000+n*4
//          其中:x=0~4，对应A~E;n=0~31
//=========================================================================
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（1）保存现场
        push {r0-r7,lr}                    //保存现场，pc(lr)入栈
        //将入口参数r1、r2转存至r2、r3，预留出r1保存引脚号
        mov r3,r2                          //r3=r2=端口引脚初始状态
        mov r2,r1                          //r2=r1=引脚方向
//-------------------------------------------------------------------------
//（2）调用内部函数，从入口参数r0中解析出端口号引脚号，分别放在r0和r1中
        bl gpio_port_pin_resolution   //调用内部解析函数,r0=端口号,r1=引脚号
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（3）获得待操作端口时钟的RCC->AHB2ENR寄存器的地址
    ldr r7,=RCC_AHB2ENR_BASE      //r7=RCC->AHB2ENR寄存器基地址
    ldr r5,[r7]                   //r5=RCC->AHB2ENR寄存器中的内容
    mov r6,#1                     //r6=1
    lsl r6,r6,r0                  //r6=待操作RCC->AHB2ENR掩码（为1的位由r0决定）
    orr r5,r6                     //或运算设GPIOxEN=1，GPIOx时钟使能
    str r5,[r7]                   //将r5中的GPIOxEN值更新到RCC->AHB2ENR寄存器中
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（4）获得待操作端口的GPIO寄存器的地址
    mov r7,r0                   //r7=r0=端口号
    ldr r4,=0x400               //r4=各端口基地址差值(0x400)
    mul r7,r7,r4                //r7=待操作端口与A口的偏移地址
    ldr r4,=GPIO_BASE           //r4=端口GPIOA基地址(即GPIO_BASE)
    add r7,r4                   //r7=待操作端口GPIO寄存器的地址
    mov r4,#2                   //r4=2，模式寄存器2位设置一个引脚模式
    mov r6,r1                   //r6=r1=引脚号
    mul r6,r6,r4                //r6=2*r1=待操作引脚左移量(GPIO_MODER_MODE_Pos)
    mov r4,#3                   //r4=3=GPIO_MODER_MODE_Msk
    lsl r4,r6                   //r4=待操作GPIO_MODER补码（为1的位由r1决定）
    mvn r4,r4                   //r4取反
    ldr r5,[r7]                 //r5=GPIO_MODER寄存器中的内容
    and r5,r4                   //与运算设~GPIOx_MODER，GPIOx_MODER清0
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（5）根据入口参数r2设定引脚输入输出状态
        cmp r2,#1                          //判断入口参数r2的值
        bne gpio_init_2          //若要设置为输入，转到gpio_init_2，将GPIOx_MODER
相应位为00
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（5.1）若要设置为输出，继续执行，将GPIOx_MODER相应位为01
    mov r4,#1                          //r4=1
    lsl r4,r4,r6                       //r4=待操作GPIO_MODER掩码（为1的位由r1决定）
    orr r5,r4                          //或运算设GPIOx_MODER=01，引脚被配置为GPIO输出功能
    str r5,[r7]                        //将r5中的GPIOx_MODER值更新到该寄存器中
    cmp r3,#1                          //判断引脚初始状态
    bne gpio_init_1                    //若为低电平，转到gpio_init_1，将BRR相应位置1
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（5.1.1）若为高电平，继续执行，将BSRR相应位置1
    mov r5,r7                    //r5=r7=待操作端口GPIO寄存器的地址
    add r5,#0x18                 //r5=待操作端口GPIO->BSRR寄存器的地址
    ldr r4,[r5]                  //r4=待操作端口GPIO->BSRR寄存器中的内容
    mov r6,#1                    //r6=1
    lsl r6,r6,r1                 //r6=待操作GPIO_BSRR掩码（为1的位由r1决定）
    orr r4,r6                    //或运算设GPIOx_BSRR=1
    str r4,[r5]                  //将r4中的GPIOx_BSRR值更新到该寄存器中
    bl  gpio_init_3              //跳转到gpio_init_3
```

补充：GPIO构件函数解析

2、gpio_init:

```
//（5.1.2）若为低电平，转到gpio_init_1，将BRR相应位置1
gpio_init_1:
        mov r5,r7                       //r5=r7=待操作端口GPIO寄存器的地址
        add r5,#0x28                    //r5=待操作端口GPIO->BRR寄存器的地址
        ldr r4,[r5]                     //r4=待操作端口GPIO->BRR寄存器中的内容
        mov r6,#1                       //r6=1
        lsl r6,r6,r1                    //r6=待操作GPIO_BRR掩码（为1的位由r1决定）
        orr r4,r6                       //或运算设GPIOx_BRR=1
        str r4,[r5]                     //将r4中的GPIOx_BRR值更新到该寄存器中
        bl  gpio_init_3
```

补充：GPIO构件函数解析

3、gpio_set:

```
//=========================================================================
//函数名称：gpio_set
//函数返回：无
// 参数说明：r0:(端口号|(引脚号))，例:(PTB_NUM|(5u))表示B口5脚，头文件中有宏定义
//          r1:希望设置的端口引脚状态（0=低电平，1=高电平）
//功能概要：当指定引脚被定义为GPIO功能且为输出时，本函数设定引脚状态
// 备    注：端口x的每个引脚控制寄存器PORTx_PCRn的地址
=PORT_PCR_BASE+x*0x1000+n*4
//          其中:x=0~4，对应A~E;n=0~31
//=========================================================================
```

补充：GPIO构件函数解析

3、gpio_set:

```
//（3）根据入口参数r3，设定引脚状态（0=低电平，1=高电平）
        cmp r3,#1                                   //判断引脚初始状态
        bne gpio_set_1          //若为低电平，转到gpio_set_1，将BRR相
应位置1
```

补充：GPIO构件函数解析

3、gpio_set:

```
//（3.1）若为高电平，继续执行，将BSRR相应位置1
    ldr r5,=GPIO_BASE+0x18          //r5=第一个端口GPIO->BSRR寄存器的地址
    add r5,r6                       //r5=待操作端口GPIO->BSRR寄存器的地址
    ldr r4,[r5]                     //r4=待操作端口GPIO->BSRR寄存器中的内容
    mov r6,#1                       //r6=1
    lsl r6,r6,r1                //r6=待操作GPIO_BSRR掩码（为1的位由r1决定）
    orr r4,r6                       //或运算设GPIOx_BSRR=1
    str r4,[r5]                     //将r4中的GPIOx_BSRR值更新到该寄存器中
    bl  gpio_set_2                  //跳转到gpio_set_2
```

补充：GPIO构件函数解析

3、gpio_set:

```
//（3.2）若为低电平，转到gpio_set_1，将GPIOx_BRR相应位置1
gpio_set_1:
        ldr r5,=GPIO_BASE+0x28          //r5=第一个端口GPIO->BRR寄存器的地址
        add r5,r6                       //r5=待操作端口GPIO->BRR寄存器的地址
        ldr r4,[r5]                     //r4=待操作端口GPIO->BRR寄存器中的内容
        mov r6,#1                       //r6=1
        lsl r6,r6,r1                    //r6=待操作GPIO_BRR掩码（为1的位由r1决定）
        orr r4,r6                       //或运算设GPIOx_BRR=1
        str r4,[r5]                     //将r4中的GPIOx_BRR值更新到该寄存器中
```

4.3.3 GPIO构件的使用方法

以控制一盏小灯闪烁为例，必须知道两点：一是由芯片的哪个引脚，二是高电平点亮还是低电平点亮。

1. 给小灯取名

在user.inc文件中给小灯起名字，并确定与MCU连接的引脚，进行宏定义。

```
.equ LIGHT_RED,(PTB_NUM|7)        //红色RUN灯使用的端口/引脚
```

2. 给小灯的亮暗取名

在user.inc文件中对小灯亮、暗进行宏定义，方便编程。

```
.equ LIGHT_ON,0                   //灯亮
.equ LIGHT_OFF,1                  //灯暗
```

3. 初始化小灯

在main.s文件中初始化小灯的初始状态为输出，并点亮

```
ldr r0,=LIGHT_RED          //r0←端口和引脚（用=是因为宏常数>=256,且用ldr)
mov r1,#GPIO_OUTPUT        //r2←引脚的为输出
mov r2,#LIGHT_ON           //r3←引脚的初始状态,默认点亮
bl  gpio_init              //调用gpio初始化函数
```

4. 点亮小灯

在main.s文件中调用gpio_set函数点亮小灯。

```
ldr   r0,=LIGHT_RED
ldr   r1,=LIGHT_ON
bl    gpio_set             //调用函数设置小灯为亮
```

实例分析：GPIO构件源码实例剖析

（1）利用Exam4_1，分析GPIO汇编构件（gpio.inc，gpio.s）

（2）利用Exam4_1_C，分析GPIO的C语言构件（gpio.h，gpio.c）

4.4 实验一：理解汇编程序框架及运行

1. 实验目的

本实验通过编程控制LED小灯，体会GPIO输出作用，可扩展控制蜂鸣器、继电器等；通过编程获取引脚状态，体会GPIO输入作用，可用于获取开关的状态。主要目的如下：

（1）了解集成开发环境的安装与基本使用方法。

（2）掌握GPIO构件基本应用方法，理解第一个汇编程序框架结构。

（3）掌握硬件系统的软件测试方法，初步理解printf输出调试的基本方法。

2. 实验准备

（1）硬件部分。PC机或笔记本电脑一台、开发套件一套。

（2）软件部分。根据电子资源“..\02-Doc”文件夹下的电子版快速指南，下载合适的电子资源。

（3）软件环境。按照电子版快速指南中“安装软件开发环境”一节，进行有关软件工具的安装。

3. 参考样例

参照“Exam4_1”工程，该样例是通过调用GPIO驱动构件方式，使得一个发光二极管闪烁。使用构件方式编程干预硬件是今后编程的基本方式，因此要充分掌握构件的使用方法。

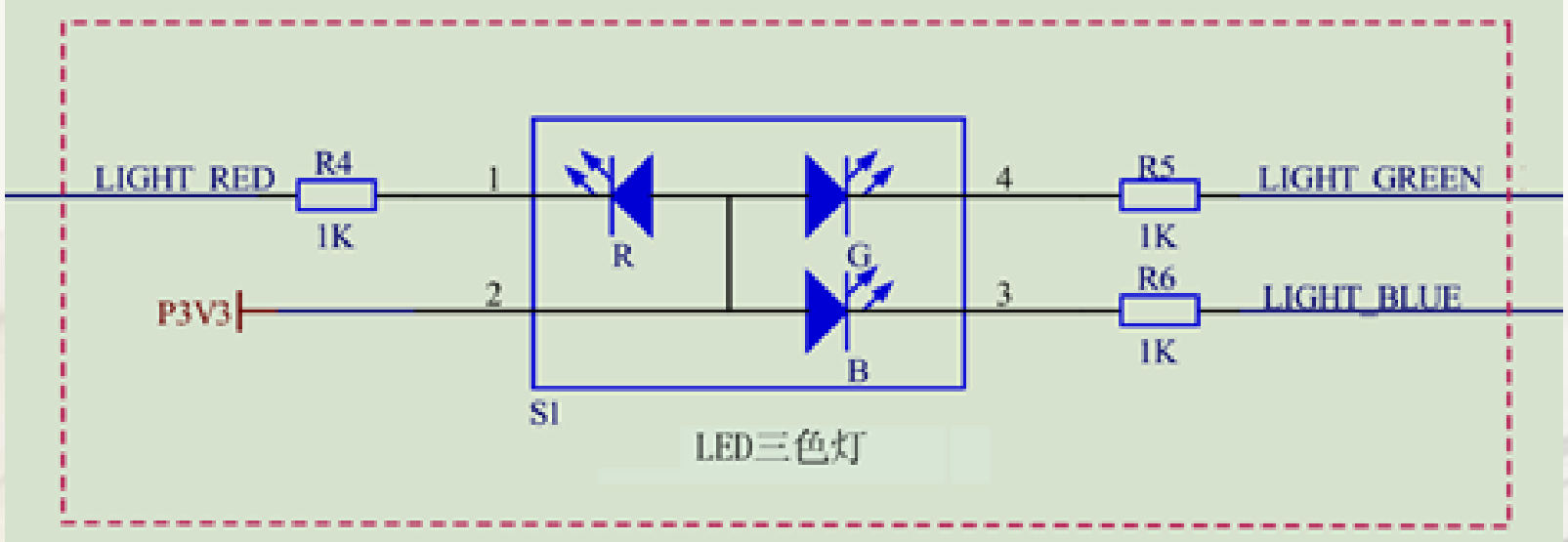
4. 实验过程或要求

（1）验证性实验

① 下载开发环境AHL-GEC-IDE。
② 建立自己的工作文件夹。
③ 拷贝模板工程并重命名。
④ 导入工程。
⑤ 编译工程。
⑥ 下载并运行。
⑦ 观察运行结果与程序的对应。

（2）设计性实验

在验证性实验的基础上，自行编程实现开发板上的红灯、蓝灯和绿灯交替闪烁.



（3）进阶实验★

对目标板上的三色灯进行编程，通过三色灯的不同组合，实现红、蓝、绿、青、紫、黄和白等灯的亮暗控制。灯颜色提示：青色为绿蓝混合，黄色为红绿混合，紫色为红蓝混合，白色为红蓝绿混合。

5. 实验报告要求

（1）基本掌握WORD文档的排版方法。
（2）用适当文字、图表描述实验过程。
（3）用200~300字写出实验体会。
（4）在实验报告中完成实践性问答题。

6. 实践性问答题

（1）比较ascii、asciz、string这三种字符串定义格式的区别。
（2）比较立即数的“#”和“=”这两个前缀的区别
（3）编写程序输出参考样例中mMainLoopCount变量的地址。
（4）集成的红绿蓝三色灯最多可以实现几种不同颜色LED灯的显示，通过实验给出组合列表。