实验二：GPIO输出

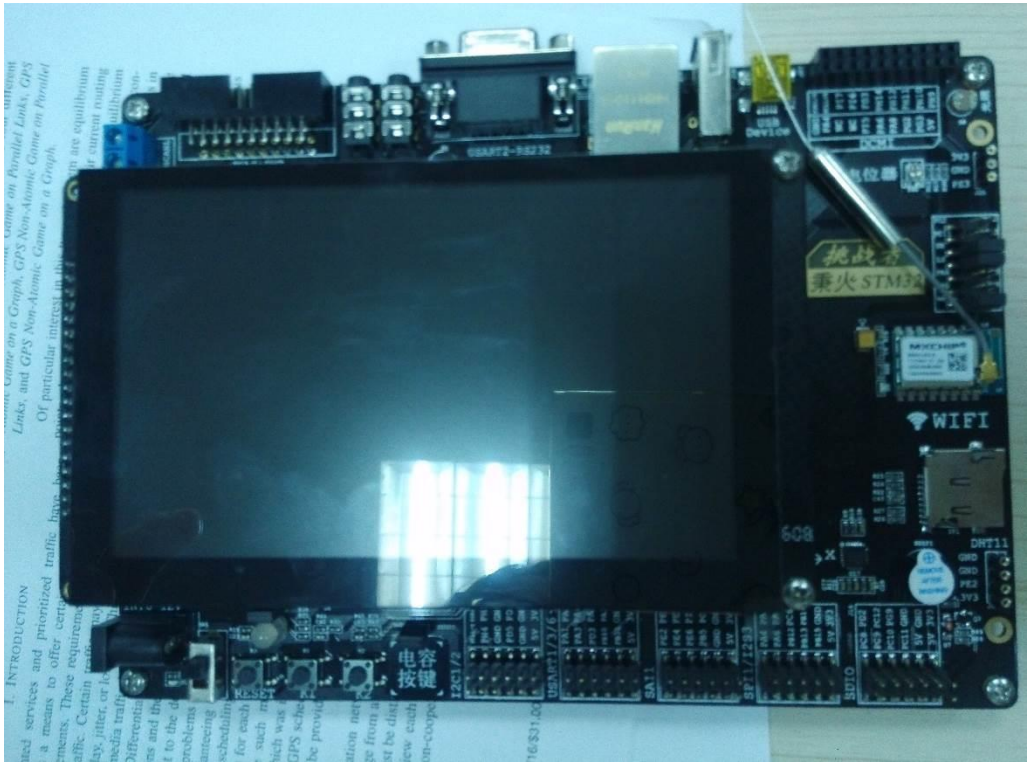
--寄存器点亮LED灯

实验目的

- 了解GPIO端与寄存器端之间的映射关系
- 学会使用寄存器点亮LED灯

实验设备

- 软件Keil5(keil提供了软件仿真功能)；
- STM32开发板



实验内容

- 了解GPIO端与寄存器端之间的映射关系
- 学会使用寄存器点亮LED灯
- 学会改变寄存器的相应位来观察LED灯的亮灭情况

0、GPIO简介

（1）GPIO简介

（2）GPIO框图讲解

（3）寄存器映射

（4）GPIO端口与寄存器

0、GPIO简介

GPIO—General Purpose Input Output

GPIO是通用输入输出端口的简称，简单来说就是软件（STM32）可控制的引脚，STM32芯片的GPIO引脚与外部设备连接起来，从而实现与外部通信、控制以及数据采集的功能。

0、GPIO简介

STM32F429IGT6 实物图

（1）芯片正面是丝印，ARM应该是表示该芯片使用的是ARM的内核，STM32F429IGT6是芯片型号，后面的字应该是跟生产批次相关，最下面的是ST的LOGO。

（2）芯片四周是引脚，左上角的小圆点表示1脚，然后从1脚起按照逆时针的顺序排列。

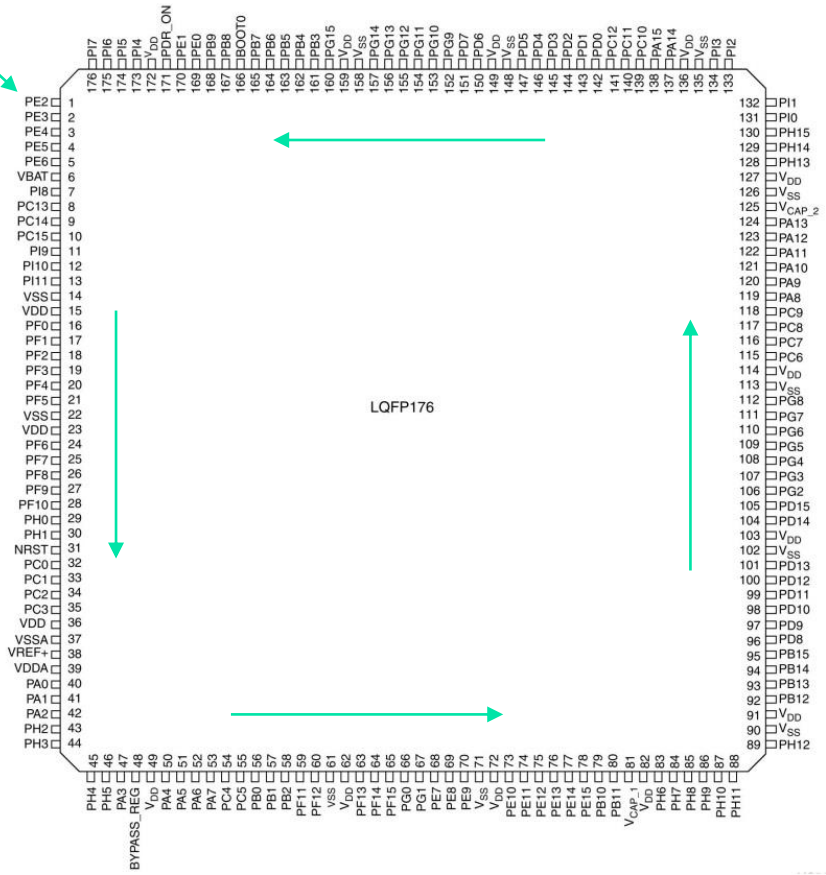
【注】开发板中把芯片的引脚引出来，连接到各种传感器上，然后在STM32上编程（实际就是通过程序控制这些引脚输出高电平或者低电平）来控制各种传感器工作。

0、GPIO简介

STM32F429IGT6正面引脚图

第一个引脚

逆时针方向到达最后一个引脚；



一共有176个引脚，其中共有144个引脚是GPIO引脚；

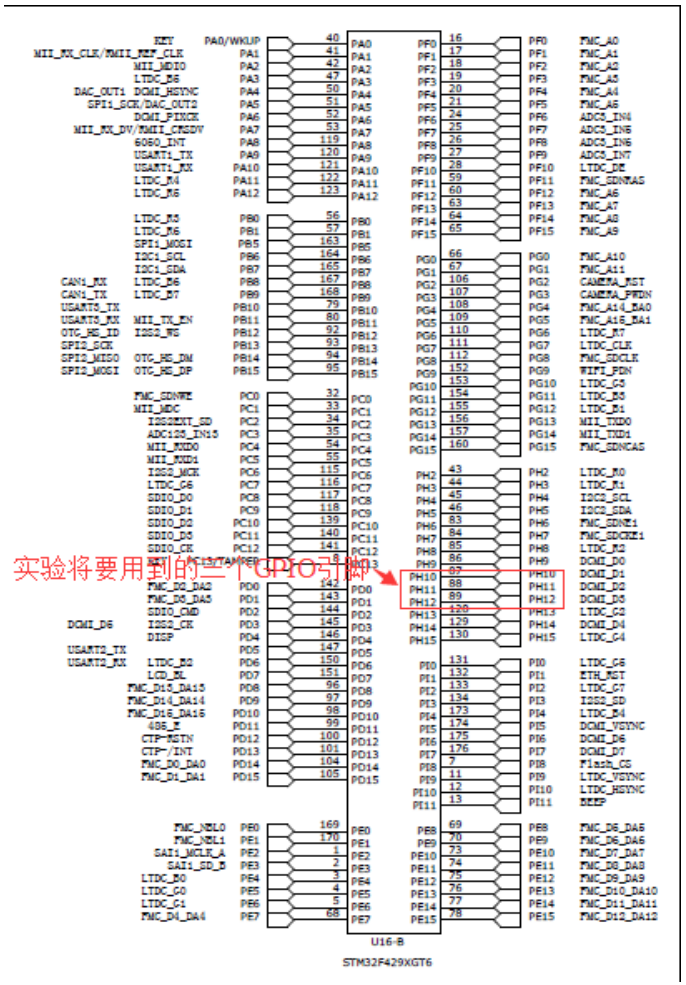
0、GPIO简介

STM32F429IGT6引脚分类：六类

引脚分类	引脚说明
电源	(VBAT)、(VDD VSS)、(VDDA VSSA)、(VREF+ VREF-)等
晶振IO	主晶振IO，RTC晶振IO
下载IO	用于JTAG下载的IO:JTMS、JTCK、JTDI、JTDO、NJTRST
BOOT IO	BOOT0、BOOT1，用于设置系统的启动方式
复位 IO	NRST，用于外部复位
上面五部分IO组成的系统我们也叫做最小系统	
GPIO	专用器件接到专用的总线，比如I2C,SPI,SDIO,FSMC,DCMI这些总线的器件需要接到专用的IO
	普通的元器件接到GPIO,比如蜂鸣器，LED，按键等元器件用普通的GPIO
	如果还有剩下的IO,可根据项目需要引出或者不引出

0、GPIO简介

- 右图表示GPIO的引脚图，STM32芯片的GPIO被分成9组
- PA,PB,PC,PD,PE,PF,PG,PH,PI，每组有16个引脚，所以共有144个GPIO引脚。
- 每组GPIO通过一组寄存器控制



图：GPIO引脚

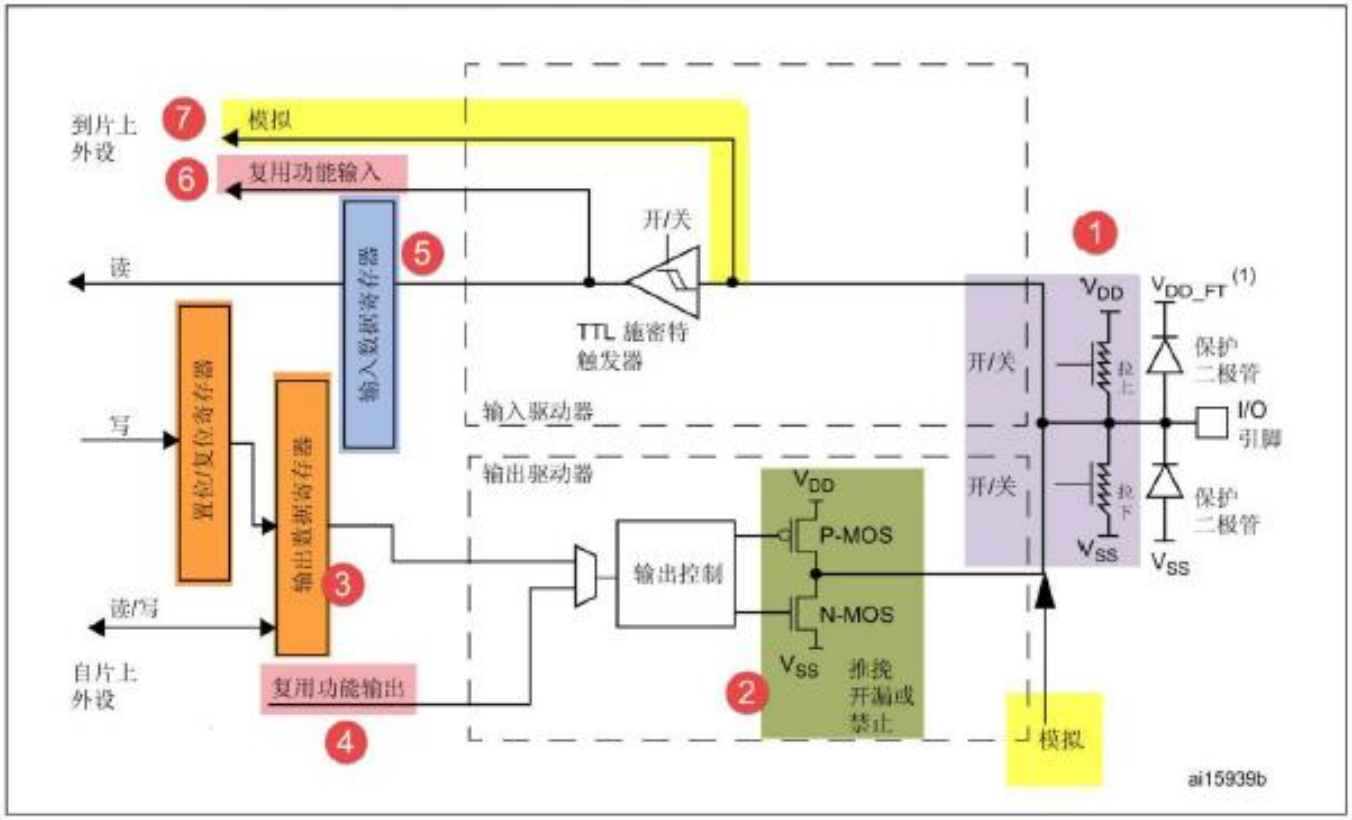
0、GPIO简介

最基本的**输出功能**是由STM32控制引脚输出高、低电平，实现开关控制，如把GPIO引脚接入到LED灯，那就可以控制LED灯的亮灭。

最基本的**输入功能**是检测外部输入电平，如把GPIO引脚连接到按键，通过电平高低区分按键是否被按下。

0、GPIO框图讲解

GPIO结构框图



通过GPIO硬件结构框图，就可以从整体上深入了解GPIO外设及它的各种应用模式。该图从最右端看起，最右端就是代表STM32芯片引出的GPIO引脚，其余部件都位于芯片内部。

0、GPIO端口与寄存器

每个通用 I/O 端口包括10个寄存器：

- 4 个 32 位配置寄存器（GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR 和 GPIOx_PUPDR）
- 2 个 32 位数据寄存器（GPIOx_IDR 和 GPIOx_ODR）
- 1 个 32 位置位/复位寄存器 (GPIOx_BSRR)
- 1 个 32 位锁定寄存器 (GPIOx_LCKR)
- 2 个 32 位复用功能选择寄存器（GPIOx_AFRH 和 GPIOx_AFRL）

0、GPIO框图讲解

GPIO工作模式

- 通过对GPIO寄存器写入不同的参数，就可以改变GPIO的应用模式。

寄存器名称	功能
模式寄存器GPIOx_MODER	配置GPIO的输入/输出/复用/模拟模式
输出类型寄存器GPIOx_OTYPER	配置推挽/开漏模式
输出速度寄存器GPIOx_OSPEEDR	配置可选2/25/50/100MHz输出速度
上/下拉寄存器GPIOx_PUPDR	配置上拉/下拉/浮空模式

0、GPIO框图讲解

GPIO工作模式

模式寄存器的MODER位[0:1]	输出类型寄存器的OTYPER位	输出速度寄存器的OSPEEDR	上/下拉寄存器的PUPDR位[0:1]
01 -输出模式	0 -推挽模式 1 -开漏模式	00 -速度2MHz 01 -速度25MHz 10 -速度50MHz 11 -速度100MHz	00 -无上拉无下拉 01 -上拉 10 -下拉 11 -保留
10 -复用模式			
00 -输入模式	不可用	不可用	
11 -模拟功能	不可用	不可用	00 -无上拉无下拉 01 -保留 10 -保留 11 -保留

表 ： GPIO寄存器的参数配置

0、GPIO框图讲解

基本结构分析

输出数据寄存器

通过修改GPIO“输出数据寄存器GPIOx_ODR” 的值就可以修改GPIO引脚的输出电平。

“置位/复位寄存器GPIOx_BSRR”可以通过修改输出数据寄存器的值从而影响电路的输出。

复用功能输出

“复用功能输出”中的“复用”是指STM32的其它片上外设对GPIO引脚进行控制。

0、GPIO框图讲解

基本结构分析

输入数据寄存器

看信号存储在“输入数据寄存器GPIOx_IDR”中，通过读取该寄存器就可以了解GPIO引脚的电平状态。

复用功能输入

与“复用功能输出”模式类似，在“复用功能输入”模式时，GPIO引脚的信号受外设控制。

0、GPIO框图讲解

基本结构分析

模拟输入输出

- 当GPIO引脚用于ADC采集电压的输入通道时，用作“模拟输入”功能，所以ADC外设要采集到原始的模拟信号。
- 当GPIO引脚用于DAC作为模拟电压输出通道时，此时作为“模拟输出”功能，模拟信号直接输出到引脚。

【注】同时，当GPIO用于模拟功能时(包括输入输出)，引脚的上、下拉电阻是不起作用的，这个时候即使在寄存器配置了上拉或下拉模式，也不会影响到模拟信号的输入输出。

0、寄存器地址

外设基地址

外设名称	外设基地址	相对**AHB1**总线的地址偏移
GPIOA	0x4002 0000	0x0
GPIOB	0x4002 0400	0x0000 0400
GPIOC	0x4002 0800	0x0000 0800
GPIOD	0x4002 0C00	0x0000 0C00
GPIOE	0x4002 1000	0x0000 1000
GPIOF	0x4002 1400	0x0000 1400
GPIOG	0x4002 1800	0x0000 1800
GPIOH	0x4002 1C00	0x0000 1C00

【注】从表格看到，GPIOA的基址相对于AHB1总线的地址偏移为0，我们应该就可以猜到，AHB1总线的第一个外设就是GPIOA。

0、寄存器映射

外设寄存器

寄存器名称	寄存器地址	相对GPIOH基址的偏移
GPIOH_MODER	0x4002 1C00	0x00
GPIOH_OTYPER	0x4002 1C04	0x04
GPIOH_OSPEEDR	0x4002 1C08	0x08
GPIOH_PUPDR	0x4002 1C0C	0x0C
GPIOH_IDR	0x4002 1C10	0x10
GPIOH_ODR	0x4002 1C14	0x14
GPIOH_BSRR	0x4002 1C18	0x18
GPIOH_LCKR	0x4002 1C1C	0x1C
GPIOH_AFRL	0x4002 1C20	0x20
GPIOH_AFRH	0x4002 1C24	0x24

STM32寄存器映射

GPIOx端口数据输出寄存器ODR描述

GPIO 端口输出数据寄存器 (GPIOx_ODR) (x = A..I) 1

GPIO port output data register

偏移地址：0x14 2

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

3

位 31:16 保留，必须保持复位值。

位 15:0 **ODRy[15:0]**：端口输出数据 (Port output data) (y = 0..15)

这些位可通过软件读取和写入。

注意：对于原子置位/复位，通过写入 GPIOx_BSRR 寄存器，可分别对 ODR 位进行置位和复位 (x = A..I)。

4

STM32 寄存器映射

GPIO端口置位/复位寄存器BSRR描述

Address offset: 0x18

Reset value: 0x0000 0000

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

- 位 31:16 **BRy**：端口 x 复位为 y (Port x reset bit y) (y = 0..15)
  - 0：不会对相应的 ODRx 位执行任何操作
  - 1：对相应的 ODRx 位进行复位

- 位 15:0 **BSy**：端口 x 置位位 y (Port x set bit y) (y= 0..15)
  - 0：不会对相应的 ODRx 位执行任何操作
  - 1：对相应的 ODRx 位进行置位

注意： 如果同时对 *BSx* 和 *BRx* 置位，则 *BSx* 的优先级更高。

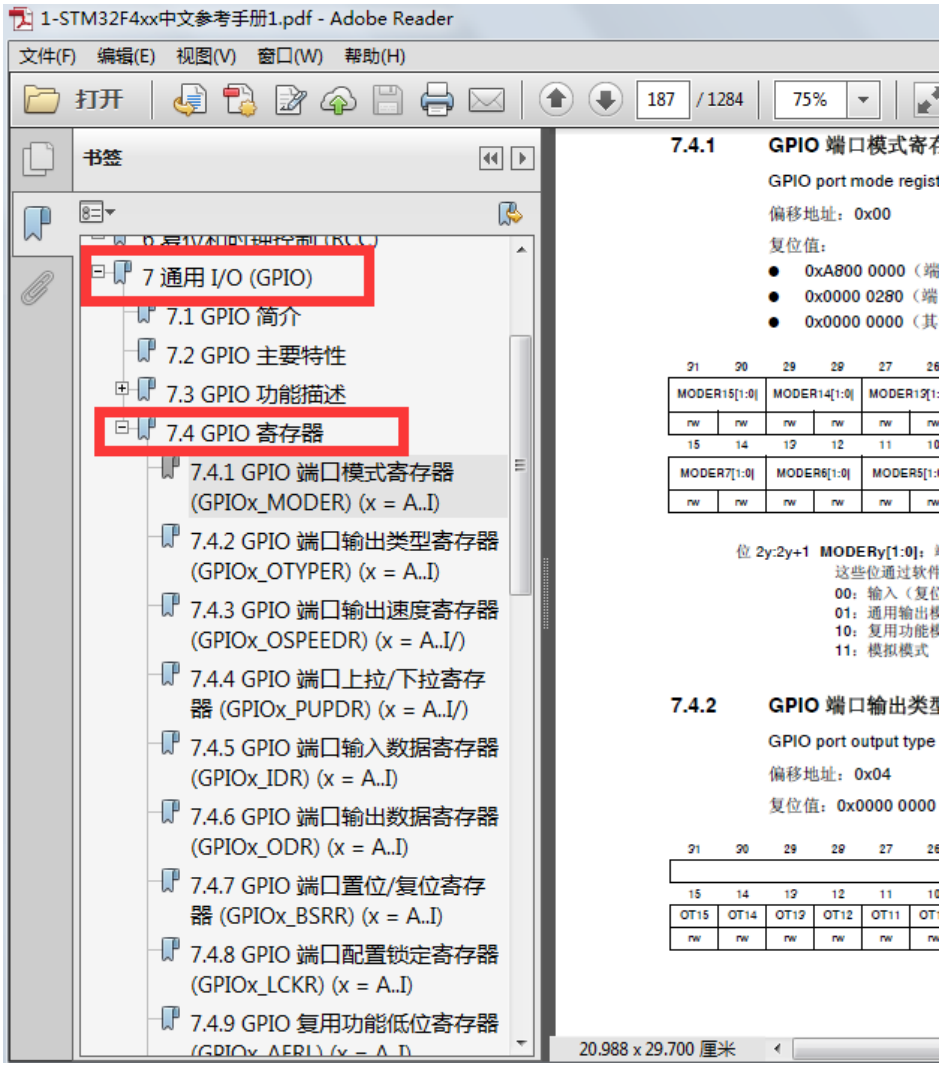
0、寄存器映射

- 寄存器映射：

给有特定功能的内存单元取一个别名，这个别名就是我们经常说的寄存器，这个给已经分配好地址的有特定功能的内存单元取别名的过程就叫寄存器映射。

0、GPIO端口与寄存器

关于其他寄存器的具体说明请参考附件中给的《1-STM32F4xx中文参考手册1》。

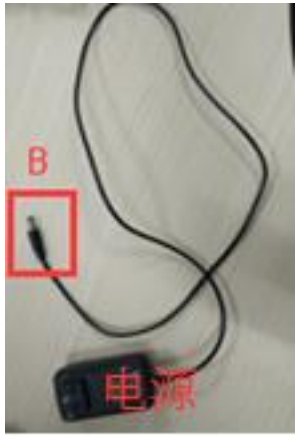


一、连接开发板

- 开发方法：宿主机/目标机的方法
- 把仿真器用USB线连接电脑，如果仿真器的灯亮表示正常，可以使用。
- 把仿真器的另外一端连接到开发板，给开发板上电。
- 通过软件KEIL给开发板下载程序。

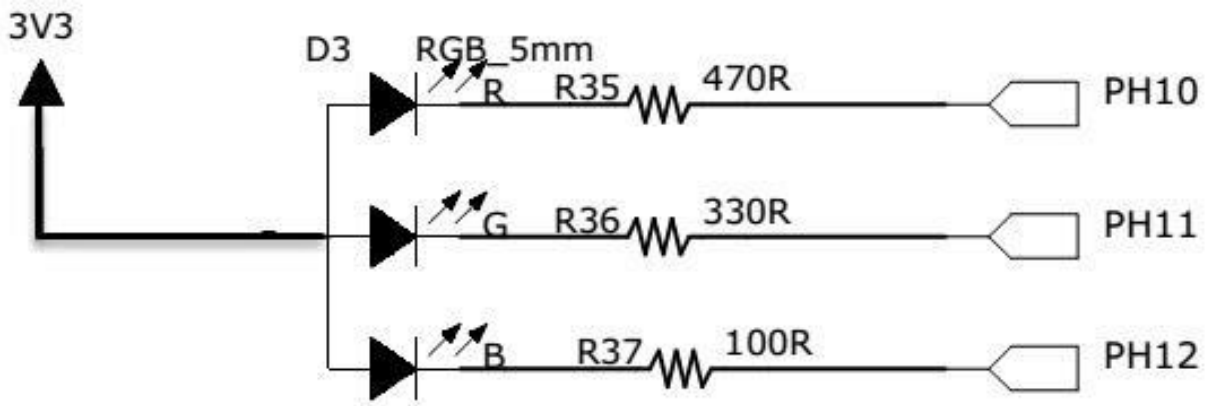
一、连接开发板

- 需要三样东西，一个开发板，一个仿真器和一个电源，如图所示
- 将仿真器的A端插入开发板的A端，注意正反面，正面的仿真器插头有长方形凸起，仿真器的另一端用USB连接电脑
- 电源的B口插入开发板的B口，另一端插入电源。C处是开发板的电源开关，拨上去后电源LED灯亮则连接正常。



二、使用寄存器点亮LED灯

1.硬件连接—STM32芯片与LED灯的连接



PH10:红色LED灯；

PH11:绿色LED灯；

PH12:蓝色LED灯；

图：LED灯电路连接图

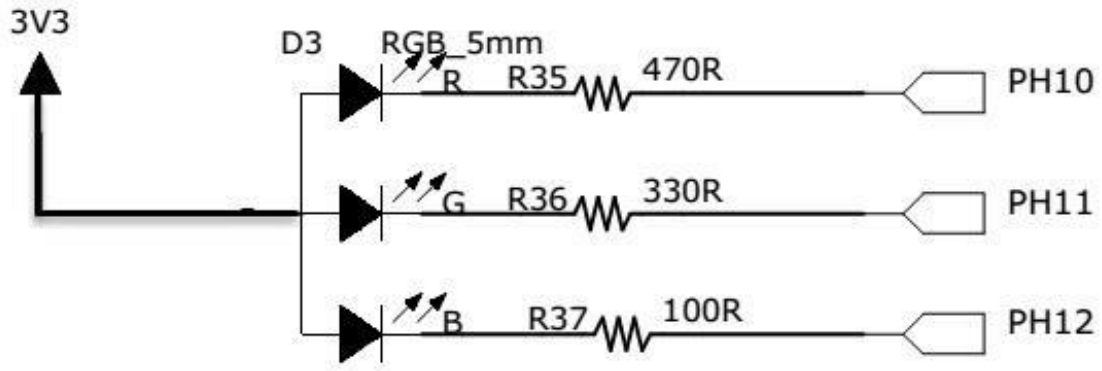
【注】：通过控制引脚输出不同颜色LED灯！

二、使用寄存器点亮LED灯

1.硬件连接—STM32芯片与LED灯的连接

图中从3个LED灯的阳极引出连接到3.3V电源，阴极各经过1个电阻引入至STM32的3个GPIO引脚PH10、PH11、PH12中，所以我们只要控制这三个引脚输出高低电平，即可控制其所连接LED灯的亮灭。

- GPIO的引脚设置成推挽输出模式(默认下拉)，输出低电平，这样就能让LED灯亮起来了。

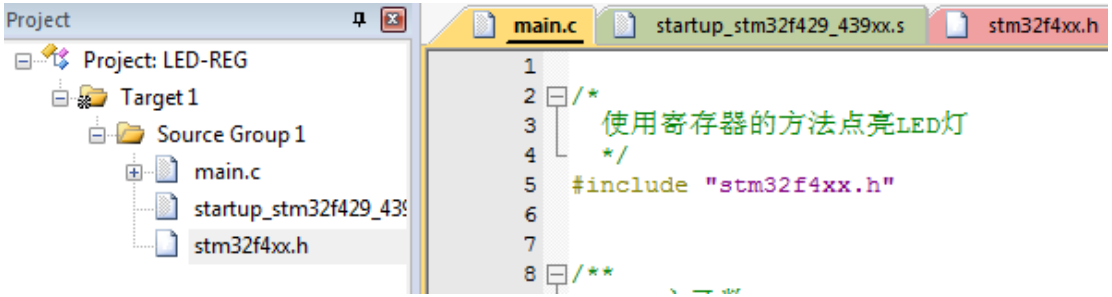


二、使用寄存器点亮LED灯

2.示例工程分析

（1）接下来，我们以示例工程（名为“示例程序1-使用寄存器点亮LED灯”）讲解如何通过控制寄存器来点亮LED灯。

（2）找到该示例工程后，在工程目录下找到后缀为“.uvprojx”的文件，用KEIL5打开即可。打开该工程，见下图，可看到一共有三个文件，分别**startup_stm32f429_439xx.s**、**stm32f4xx.h**以及**main.c**，下面我们对这三个文件进行讲解。



二、使用寄存器点亮LED灯

2.示例工程分析—启动文件

名为“startup_stm32f429_439xx.s”的文件，它里边使用汇编语言写好了基本程序，当STM32芯片上电启动的时候，首先会执行这里的汇编程序，从而建立起C语言的运行环境，所以我们把这个文件称为启动文件。

二、使用寄存器点亮LED灯

2.示例工程分析—启动文件

对于启动文件这部分，我们主要总结它以下两点功能：

（1）调用SystemInit() 函数配置STM32的系统时钟；

（2）设置C库的分支入口“__main”（最终用来调用main函数）；

【注】：相关代码见下页ppt...

二、使用寄存器点亮LED灯

2.示例工程分析—启动文件

在启动文件中有一段复位后立即执行的程序，代码如下：

```
185  ; Reset handler
186  Reset_Handler    PROC
187                   EXPORT  Reset_Handler             [WEAK]
188          IMPORT  SystemInit
189          IMPORT  __main
190
191                   LDR     R0, =SystemInit
192                   BLX     R0
193                   LDR     R0, =__main
194                   BX      R0
195                   ENDP
```

【注】：此段程序的具体解释见下页ppt...

二、使用寄存器点亮LED灯

2.示例工程分析—启动文件

（1）186行开头的是程序注释，在汇编里面注释用的是“;”；

（2）第187行是定义了一个子程序：Reset_Handler。PROC 是子程序定义伪指令。

（3）第188行 EXPORT 表示 Reset_Handler 这个子程序可供其他模块调用。关键字[WEAK]表示弱定义，如果编译器发现在别处定义了同名的函数，则在链接时用别处的地址进行链接，如果其它地方没有定义，编译器也不报错，以此处地址进行链接。

（4）第189行和第190行 IMPORT 说明 SystemInit 和__main 这两个标号在其他文件，在链接的时候需要到其他文件去寻找。

（5）第191行把 SystemInit 的地址加载到寄存器 R0。

（6）第192行程序跳转到 R0 中的地址执行程序，即执行SystemInit函数的内容。

（7）第193行把__main 的地址加载到寄存器 R0。

（8）第194行程序跳转到 R0 中的地址执行程序，即执行__main函数，执行完毕（9）之后就去到我们熟知的 C 世界，进入main函数。

（10）第195行表示子程序的结束。

二、使用寄存器点亮LED灯

2.示例工程分析—启动文件

总之，看完这段代码后，了解到如下内容即可：我们需要在外部定义一个SystemInit函数设置STM32的时钟；STM32上电后，会执行SystemInit函数，最后执行我们C语言中的main函数。

二、使用寄存器点亮LED灯

2.示例工程分析--stm32f4xx.h文件

连接LED灯的GPIO引脚，是要通过读写寄存器来控制的，所以此处我们根据STM32的存储分配先定义好各个寄存器的地址，把这些地址定义都统一写在stm32f4xx.h文件中，代码如下：

二、使用寄存器点亮LED灯

2.示例工程分析--stm32f4xx.h文件

```
1
2  /*片上外设基地址  */
3  #define PERIPH_BASE           ((unsigned int)0x40000000)
4
5  /*总线基地址 */
6  #define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000)
7
8  /*GPIO外设基地址*/
9  #define GPIOH_BASE            (AHB1PERIPH_BASE + 0x1C00)
10
11
12 /* GPIOH寄存器地址,强制转换成指针 */
13 #define GPIOH_MODER       *(unsigned int*)(GPIOH_BASE+0x00)
14 #define GPIOH_OTYPER      *(unsigned int*)(GPIOH_BASE+0x04)
15 #define GPIOH_OSPEEDR     *(unsigned int*)(GPIOH_BASE+0x08)
16 #define GPIOH_PUPDR       *(unsigned int*)(GPIOH_BASE+0x0C)
17 #define GPIOH_IDR         *(unsigned int*)(GPIOH_BASE+0x10)
18 #define GPIOH_ODR         *(unsigned int*)(GPIOH_BASE+0x14)
19 #define GPIOH_BSRR          *(unsigned int*)(GPIOH_BASE+0x18)
20 #define GPIOH_LCKR          *(unsigned int*)(GPIOH_BASE+0x1C)
21 #define GPIOH_AFRL          *(unsigned int*)(GPIOH_BASE+0x20)
22 #define GPIOH_AFRH          *(unsigned int*)(GPIOH_BASE+0x24)
23
24 /*RCC外设基地址*/
25 #define RCC_BASE              (AHB1PERIPH_BASE + 0x3800)
26
27 /*RCC的AHB1时钟使能寄存器地址,强制转换成指针*/
28 #define RCC_AHB1ENR       *(unsigned int*)(RCC_BASE+0x30)
```

（1）GPIO外设的寄存器的地址值都直接强制转换成了指针，方便使用。

（2）代码的最后两段是RCC外设寄存器的地址定义，RCC外设是用来设置时钟的，以后我们会详细分析，本实验中只要了解到使用GPIO外设必须开启它的时钟即可。(见中文手册P135）

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

接下来，我们就可以开始编写程序了，我们按照以下顺序对GPIO相应寄存器进行配置。

（1）确定引脚号（PH10，PH11，PH12）；

（2）配置GPIOH_MODER寄存器：输入模式/输出模式；

（3）配置GPIOH_OTYPER寄存器：推挽模式/开漏模式；

（4）配置GPIOH_OSPEEDR寄存器：输出速度；

（5）配置GPIOH_PUPDR寄存器：上拉模式/下拉模式；

【注】：由于此次实验验证输出功能，可以不配置GPIOH_PUPDR寄存器，但由于上拉模式会小幅度提高电流输出能力，所以我们将其配置成上拉模式。

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（1）GPIO模式

首先我们把连接到LED灯的PH10引脚配置成输出模式，即配置GPIO的MODER寄存器。MODER中包含0-15号引脚，每个引脚占用2个寄存器位。这两个寄存器位设置成“01”时即为GPIO的输出模式，代码如下：

```
18    /*GPIOH MODER10清空*/
19    GPIOH_MODER  &= ~( 0x03<< (2*10));
20    /*PH10 MODER10 = 01b 输出模式*/
21    GPIOH MODER |= (1<<2*10);
```

代码：配置输出模式

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（2）输出类型

GPIO输出有推挽和开漏两种类型，我们了解到开漏类型不能直接输出高电平，要输出高电平还要在芯片外部接上拉电阻，不符合我们的硬件设计，所以我们直接使用推挽模式。配置OTYPER寄存中的OTYPER10寄存器位，该位设置为0时PH10引脚即为推挽模式，代码如下：

```
23    /*GPIOH OTYPER10清空*/
24    GPIOH_OTYPER &= ~(1<<1*10);
25    /*PH10 OTYPER10 = 0b 推挽模式*/
26    GPIOH_OTYPER |= (0<<1*10);
```

代码：设置为推挽模式

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（3）输出速度

GPIO引脚的输出速度是引脚支持高低电平切换的最高频率，本实验可以随便设置。此处我们配置OSPEEDR寄存器中的寄存器位OSPEEDR10即可控制PH10的输出速度，代码如下：

```
28    /*GPIOH OSPEEDR10清空*/
29    GPIOH_OSPEEDR &= ~(0x03<<2*10);
30    /*PH10 OSPEEDR10 = 0b 速率2MHz*/
31    GPIOH_OSPEEDR |= (0<<2*10);
```

代码：设置输出速度为2MHz

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（4）上/下拉模式

我们的GPIO引脚用于输出，引脚受ODR寄存器影响，ODR寄存器对应引脚位初始初始化后默认值为0，引脚输出低电平，所以这时我们配置上/下拉模式都不会影响引脚电平状态。但因此处上拉能小幅提高电流输出能力，我们配置它为上拉模式，即配置PUPDR寄存器的PUPDR10位，设置为二进制值“01”，代码如下：

```
33    /*GPIOH PUPDR10清空*/
34    GPIOH_PUPDR &= ~(0x03<<2*10);
35    /*PH10 PUPDR10 = 01b 上拉模式*/
36    GPIOH_PUPDR |= (1<<2*10);
```

代码：设置为下拉模式

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（5）控制引脚输出电平

在输出模式时，对BSRR寄存器或ODR寄存器写入参数即可控制引脚的电平状态。简单起见，此处我们使用BSRR寄存器控制，对相应的BR10位设置为1时PH10即为低电平，点亮LED灯，对它的BS10位设置为1时PH10即为高电平，关闭LED灯，代码如下：

```
38    /*PH10 BSRR寄存器的 BR10置1，使引脚输出低电平*/
39    GPIOH_BSRR |= (1<<16<<10);
40
41    /*PH10 BSRR寄存器的 BS10置1，使引脚输出高电平*/
42    GPIOH_BSRR |= (1<<10);
```

代码：控制引脚输出电平

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（5′）控制引脚输出电平

在输出模式时，对BSRR寄存器或ODR寄存器写入参数即可控制引脚的电平状态。对相应的ODR寄存器的相应位位设置为0时 PH10即为低电平，点亮LED灯，对它的ODR10位设置为1时PH10即为高电平，关闭LED灯，代码如下：

```
54    /*GPIOH ODR10清空*/
55    GPIOH_ODR &= ~(1<<1*10);
56    /*PH10 ODR10 = 0b */
57    GPIOH_ODR |= (0<<1*10);
```

ODR10设置为0，点亮LED灯

```
59    /*GPIOH ODR10清空*/
60    GPIOH_ODR &= ~(1<<1*10);
61    /*PH10 ODR10 = 1b */
62    GPIOH_ODR |= (1<<1*10);
```

ODR10设置为1，点亮LED灭

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

（6）开启外设时钟

如果想要外设工作，必须把相应的时钟打开。STM32 的所有外设的时钟由一个专门的外设来管理，叫 RCC（reset and clockcontrol），所有的 GPIO都挂载到 AHB1 总线上，所以它们的时钟由AHB1外设时钟使能寄存器(RCC_AHB1ENR)来控制，其中 GPIOH 端口的时钟由该寄存器的位 7 写 1 使能，开启GPIOH端口时钟，代码如下：

```
13  /*开启 GPIOH 时钟，使用外设时都要先开启它的时钟*/
14  RCC_AHB1ENR |= (1<<7);
```

代码：开启端口时钟

二、使用寄存器点亮LED灯

2.示例工程分析—main.c文件

完整代码见文档！

二、使用寄存器点亮LED灯

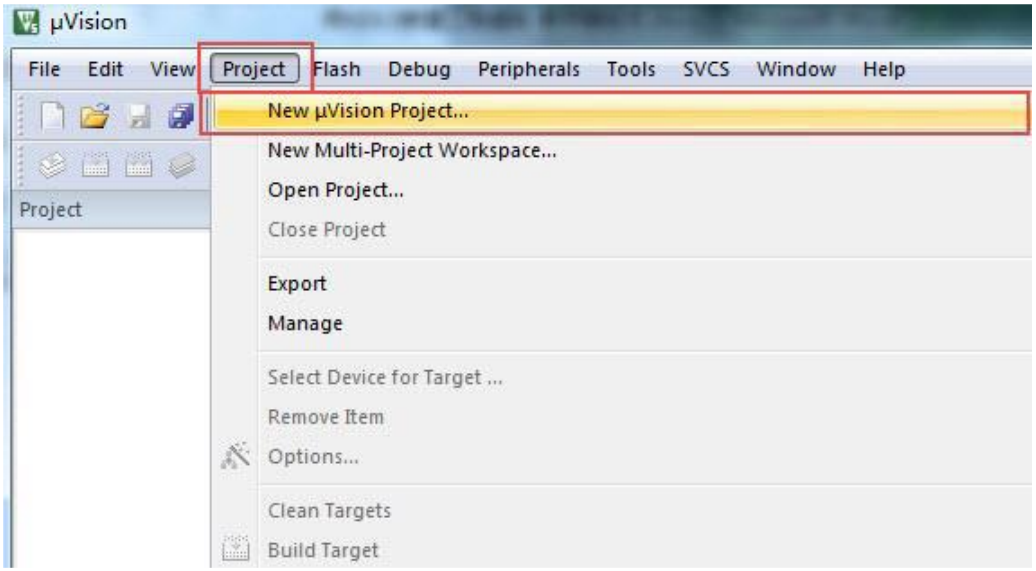
2.示例工程分析

开启时钟，配置引脚模式，控制电平，经过这三步，我们总算可以控制一个 LED了。程序编写好并且编译通过之后，如何将程序下载到开发板上来观察LED灯呢？

二、使用寄存器点亮LED灯

3.新工程创建

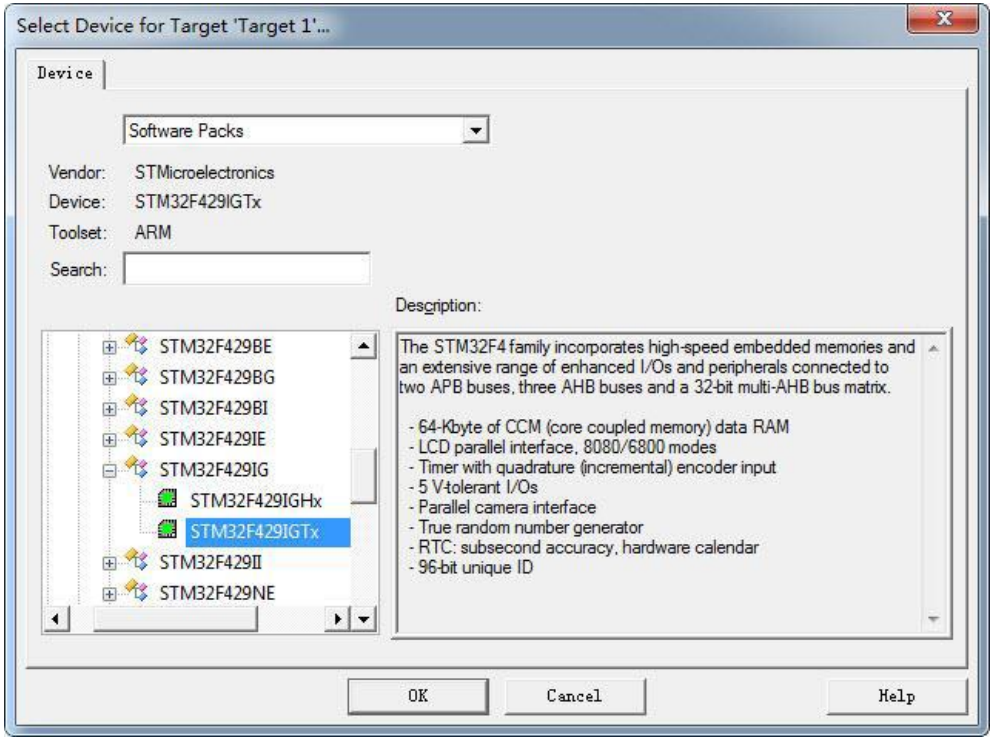
- 新建本地工程文件夹，如LED
- 新建工程



二、使用寄存器点亮LED灯

2.新工程创建

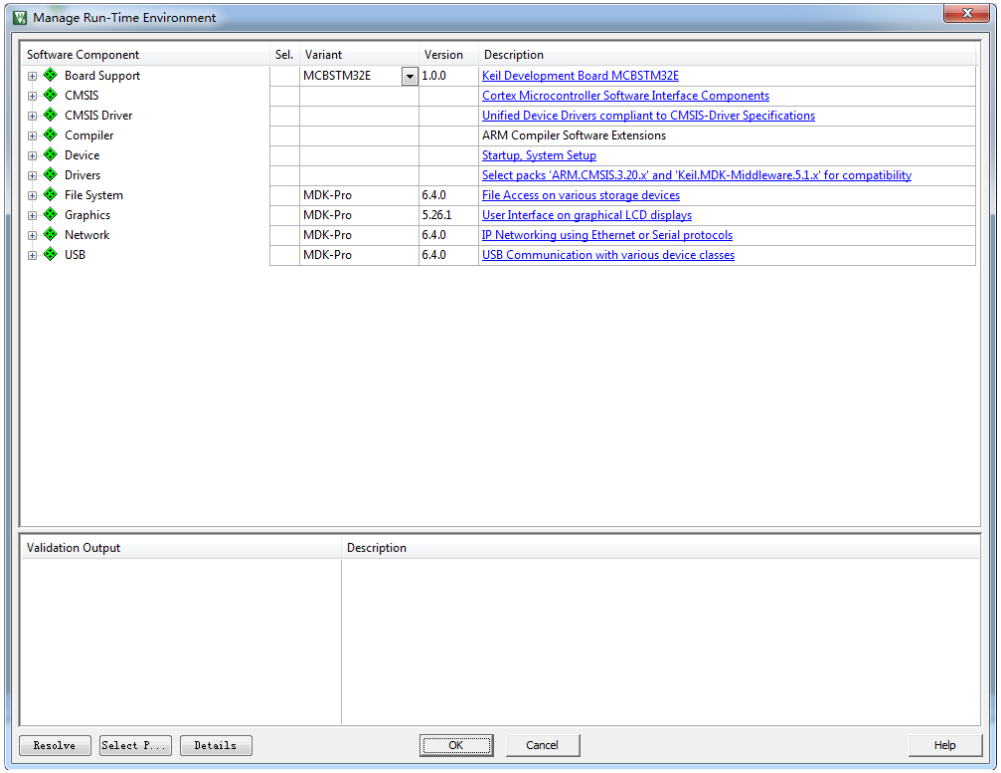
- 新建工程
  - 选择CPU型号



二、使用寄存器点亮LED灯

2.新工程创建

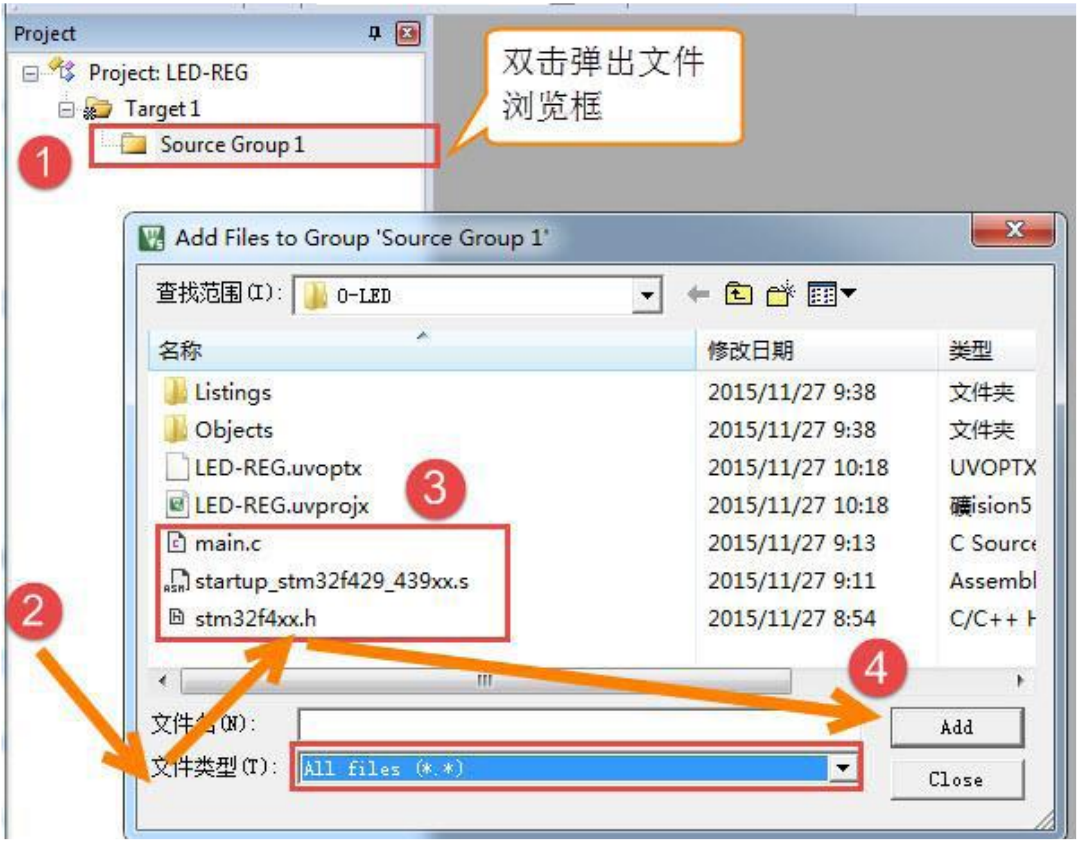
- 新建工程
  - 选择CPU型号
  - 在线添加库文件（略，直接OK）



二、使用寄存器点亮LED灯

2.新工程创建

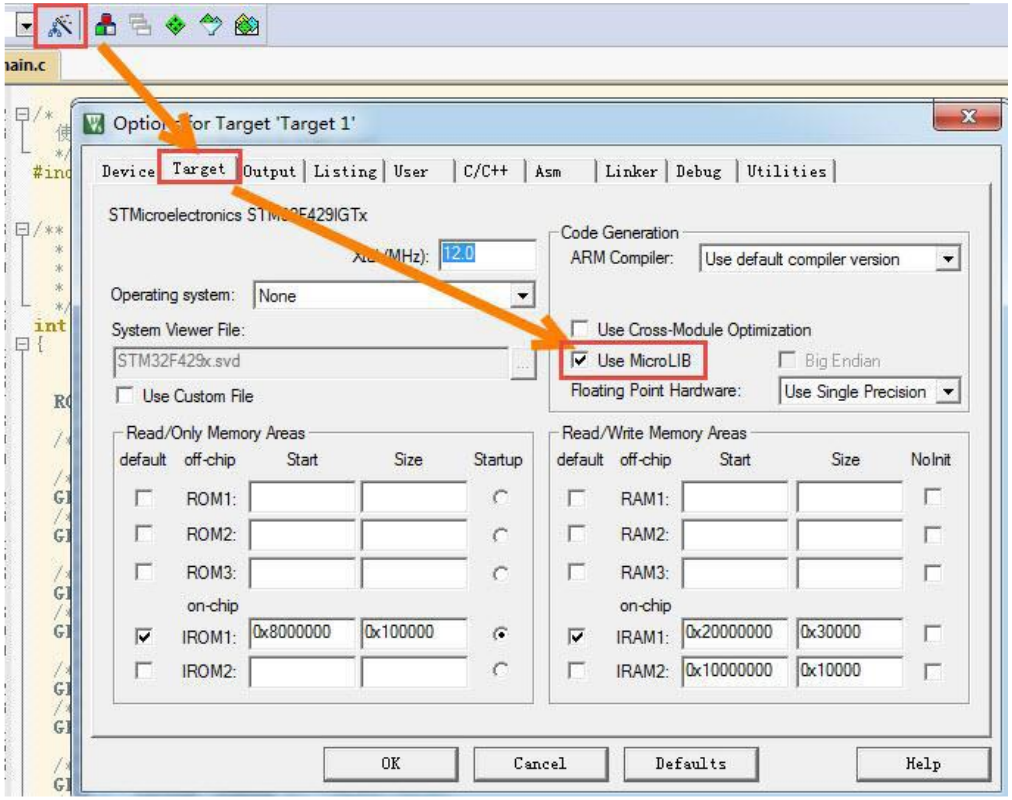
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件



二、使用寄存器点亮LED灯

2.新工程创建

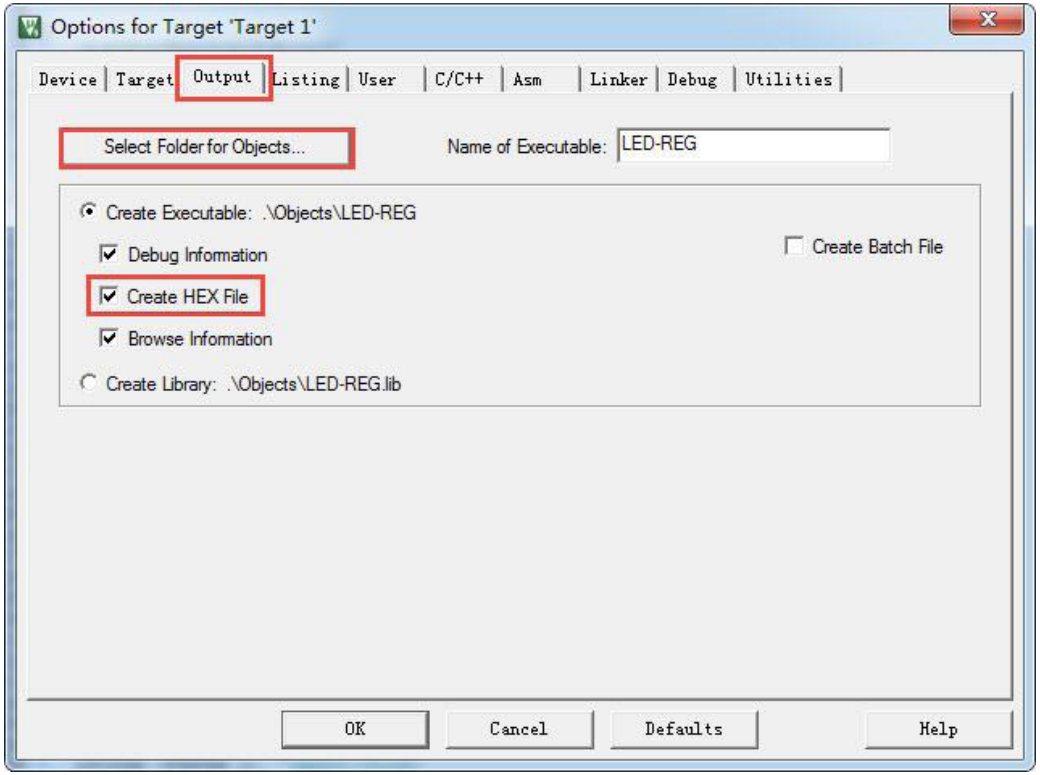
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

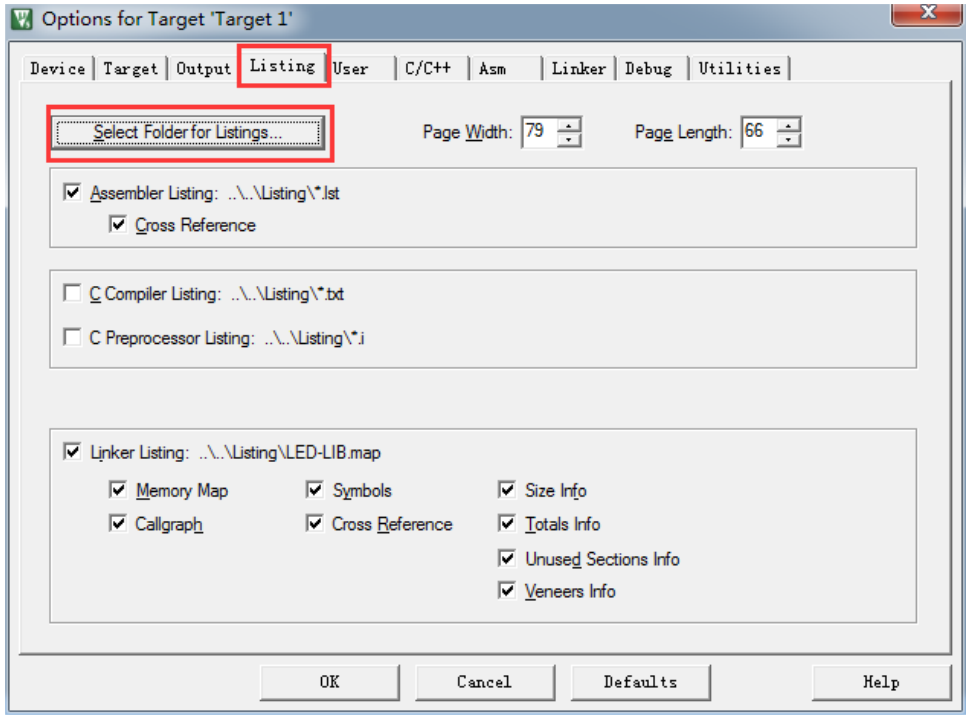
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

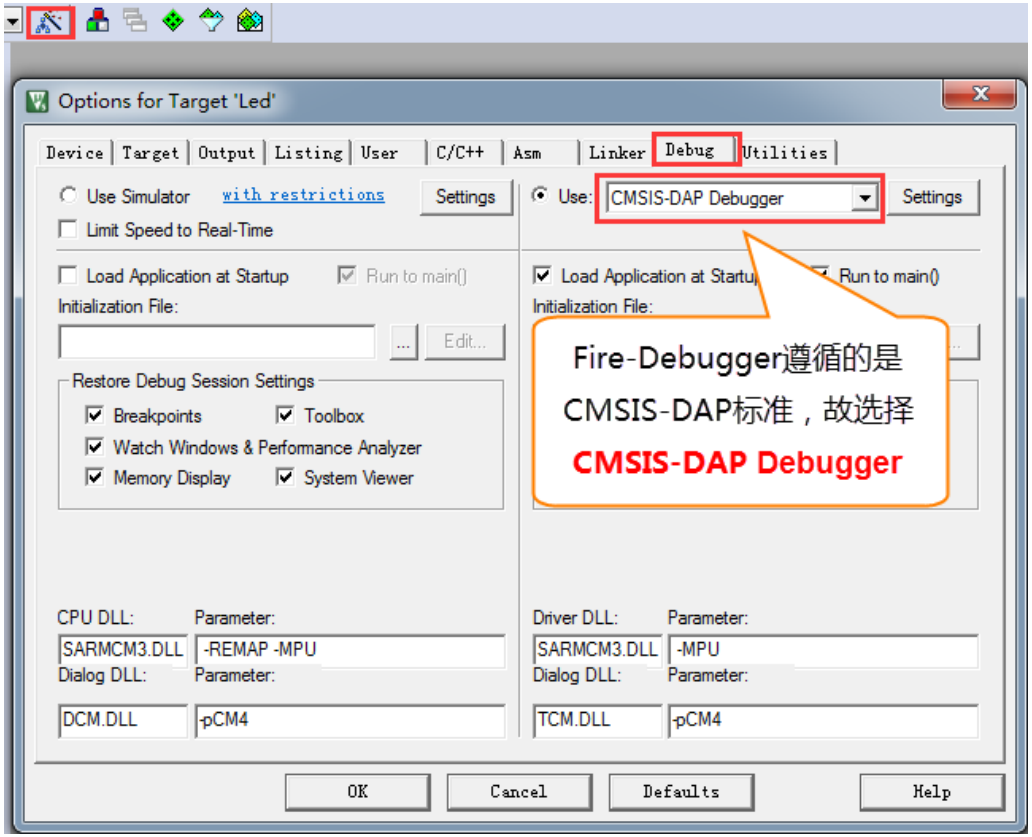
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

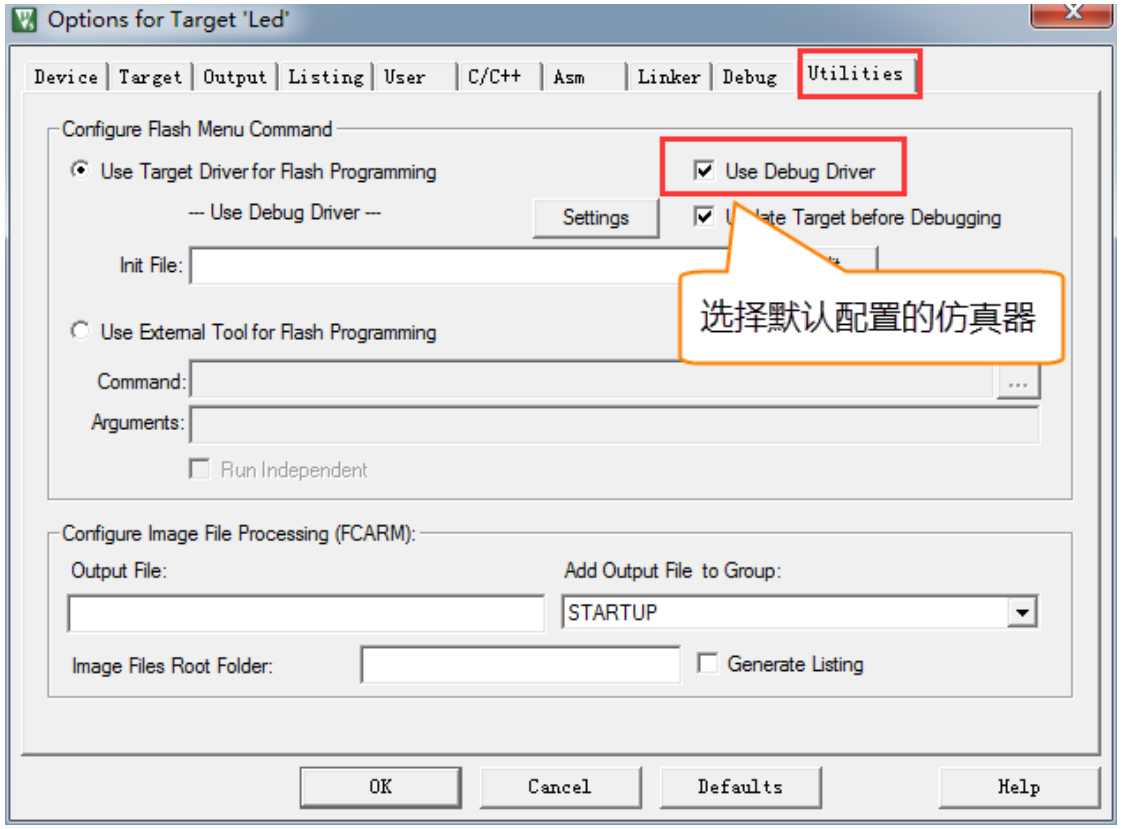
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

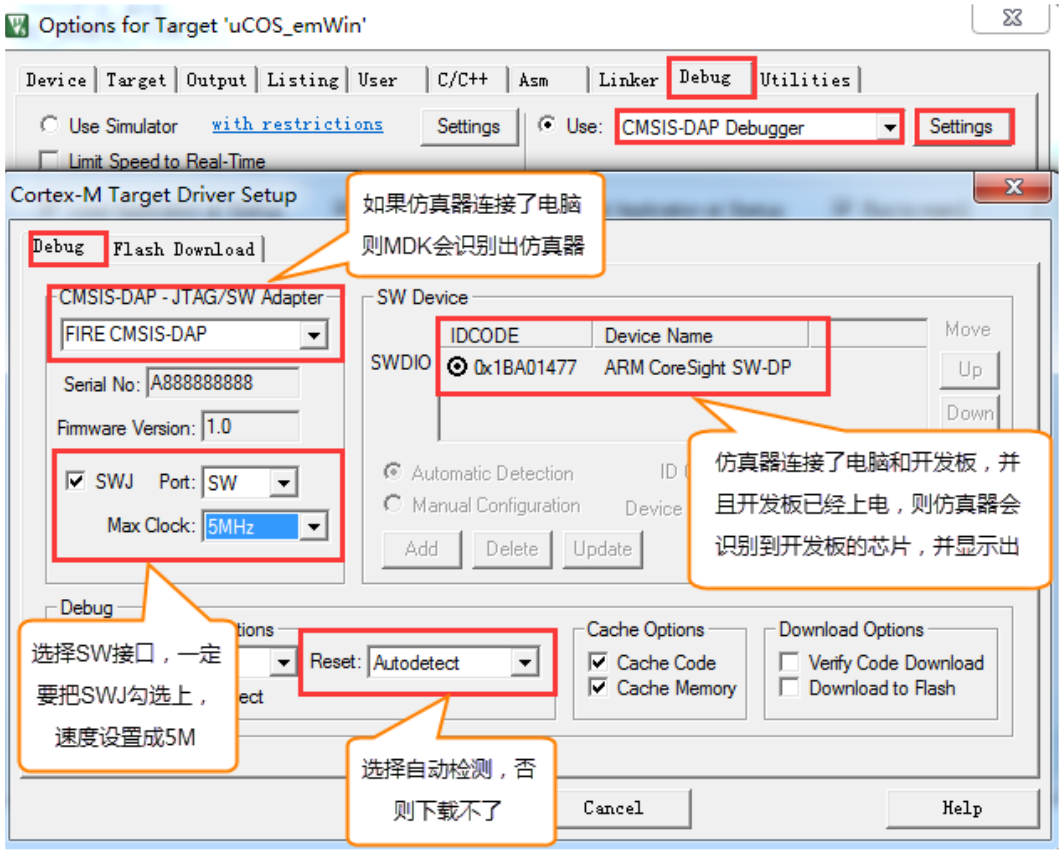
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

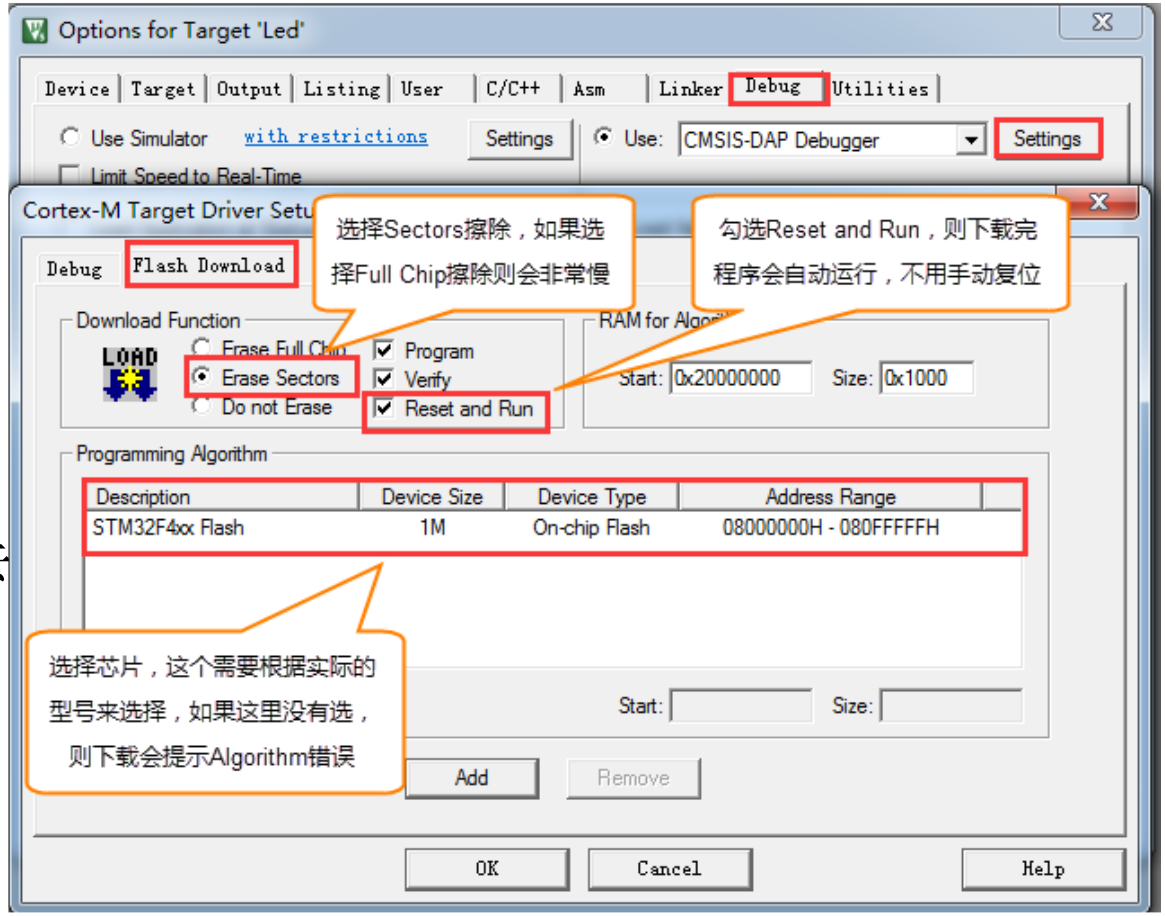
- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

2.新工程创建

- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡



二、使用寄存器点亮LED灯

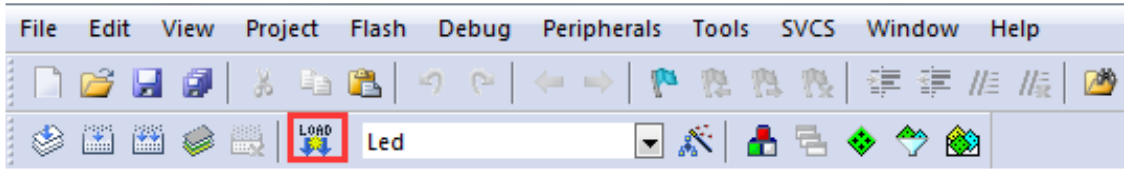
2.新工程创建

- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡
  - 编译程序 

二、使用寄存器点亮LED灯

2.新工程创建

- 新建工程
  - 选择CPU型号
  - 在线添加库文件
  - 添加文件
  - 配置魔术棒选项卡
  - 编译程序
  - 下载程序
    - 程序下载后，Build Output 选项卡如果打印出 Application running...则表示程序下载成功。
    - 如果没有出现实验现象，按复位键试试。



三、实验题目

仔细研究示例程序1的代码，掌握每个寄存器的作用，并在此基础上，对源代码进行修改，完成以下几个小作业，并进行仿真，在此基础上，**完成实验报告。**

（1）验证示例程序1实验结果是红灯亮，

（2）修改程序使得最终实验结果为蓝灯亮；

（3）修改程序使得蓝灯和绿灯同时亮，观察此时LED灯的颜色，并记录；

（4）完成“跑马灯”实验，使得LED灯按照“红-绿-蓝”的规则次序滚动点亮；