第二讲 点亮 LED

一、配置

具体配置和连接按照第一讲中进行。由实验板上的标识可以看出，LED 共有两个，LED1 连接 P1.0，LED2 连接 P1.6

二、程序结构

```
#include "io430.h"

int main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  return 0;
}
```

这是软件初始生成的程序

其引入的头文件 io430.h 是一个通用头文件，它会根据所选用的 device 的不同，来调取不同芯片的具体头文件。

打开 io430.h，即可看到

```
 ************************************************************

/************************************************************
 *  msp430x1xx family
 ************************************************************
#ifndef __io430
#define __io430


#ifndef _SYSTEM_BUILD
#pragma system_include
#endif

#if defined (__MSP430C111__) || defined (__MSP430C112__)
#include "io430x11x.h"

#elif defined (__MSP430F122__) || defined (__MSP430F123__
#include "io430x12x.h"

#elif defined (__MSP430C1111__) || defined (__MSP430C1121
#include "io430x11x1.h"
```

它会根据 define 内容不同，加入其他头文件

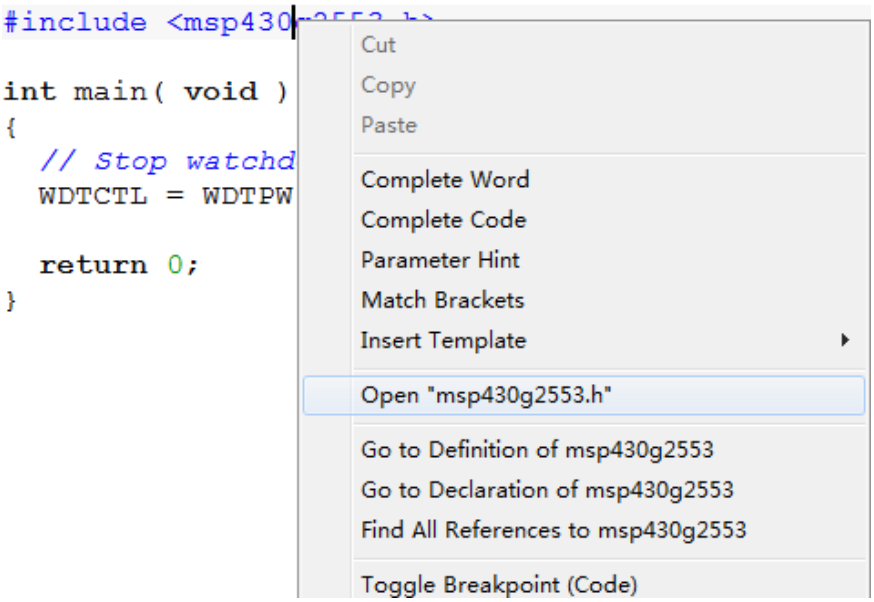
为方便使用和学习，我们直接使用 msp430g2553.h

```
#include <msp430g2553.h>

int main( void )
{
  // Stop watchdog timer to preven
  WDTCTL = WDTPW + WDTHOLD;

  return 0;
}
```

由此，我们即可随时打开 msp430g2553.h，查看此芯片相关寄存器可配置内容，方便学习和使用。



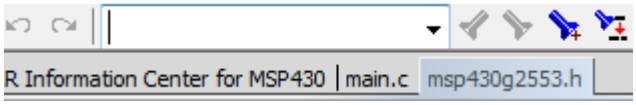
从下图所示的标准比特的定义中，我们可以看到

```
/*****************************************
* STANDARD BITS
*****************************************

#define BIT0                (0x0001u)
#define BIT1                (0x0002u)
#define BIT2                (0x0004u)
#define BIT3                (0x0008u)
#define BIT4                (0x0010u)
#define BIT5                (0x0020u)
#define BIT6                (0x0040u)
#define BIT7                (0x0080u)
#define BIT8                (0x0100u)
#define BIT9                (0x0200u)
#define BITA                (0x0400u)
#define BITB                (0x0800u)
#define BITC                (0x1000u)
#define BITD                (0x2000u)
#define BITE                (0x4000u)
#define BITF                (0x8000u)
```

因此，我们在程序编写的过程中，可以使用 BITx 来对每一位 io 进行操作。

另外，遇到不明白含义的定义，可以在头文件中搜索



好了，这个先讲到这里，我们继续看程序。
这是程序中第一个语句，刚刚接触的同学确实不明白它的含义

```
// Stop watchdog timer to prevent time out reset
WDTCTL = WDTPW + WDTHOLD;
```

这时候，就在头文件里面搜索一下吧

```
#define WDTCTL_              (0x0120u)  /* Watchdog Timer Control */
DEFW(    WDTCTL             , WDTCTL_)
/* The bit names have been prefixed with "WDT" */
```

根据头文件的说明，这是看门狗定时器。
在由单片机构成的微型计算机系统中，由于单片机的工作常常会受到来自外界电磁场的干扰，造成程序的跑飞，而陷入死循环，程序的正常运行被打断，由单片机控制的系统无法继续工作，会造成整个系统的陷入停滞状态，发生不可预料的后果，所以出于对单片机运行状态进行实时监测的考虑，就有了看门狗。
由于我们此时不需要使用，就先把它关掉。因为它的默认设置中，程序开始执行到一定时间就会自动复位，会影响我们的使用。
接下来

```
return 0;
```

到此，程序结束。

三、I/O 介绍

I/O，是 INPUT/OUTPUT 的简写，即输入输出。我们使用的 MSP430G2553 共有两组 I/O 口，即 P1.0-P1.7，P2.0-P2.5。如对于 P1，它的输入输出值分别存储在 P1IN 和 P1OUT 两个 8 位寄存器中，其中 P1.0 对应最低位，P1.7 对应最高位。且对于 8 位寄存器，仅最高位为 1 时其值为 0x80，仅最低位为 1 时其值为 0x01。

当然，同一个 I/O 口不能同时进行输入、输出，每一时刻只能进行一项工作，并由 P1DIR 寄存器控制，其中 1 为输出，0 为输入。P1DIR 默认值为 0x00，即全为输入。使用时，可以采用头文件中的宏定义来增强可读性，如 P1DIR=BIT0+BIT4，即将 P1.0,P1.4 设置为输出，其他位输入。

需要注意的是，msp430 系列单片机不支持位操作，即不能操作如 P1^2=1。因此，为了仅对个别位进行操作时，采用|=，&=，^=。
如 P1OUT|=BIT3; 将 P1.3 置 1，而其他位不变
P1OUT&=~BIT2; 将 P1.2 置 0，其他位不变
P1OUT^=BIT4; 将 P1.4 取反，其他位不变
当然，也可以直接计算所需的值，如我们需要第 0 位和第三位为高，其余为低，可以直接赋值 P1OUT=0x09;这里 0x09 是 16 进制表示，对应的 2 进制为 00001001。

四、点亮 LED

我们进入正题，点亮 LED。在此实验板上，LED1 的正极连接到了 P1.0，负极接 470Ω 电

阻接地。因此，要点亮 LED1，就要给 P1.0 置为高电平。（在其他场合，要根据具体连接方式，不能认为 P1.0 都是 LED，也不能认为 LED 都是置 1 才会亮）

修改程序如下：

```
#include <msp430g2553.h>

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  P1DIR = BIT0;//P1.0设置为输出
  P1OUT = BIT0;//P1.0输出高电平

  while(1);

}
```

为了在程序运行后不会复位，在最后加入死循环 while(1)，使其停在这里。

另外，去掉 return，并将 main 改为 void 类型，以消除 warning

```
Messages
Building configuration: 2 - Debug
Updating build tree...
main.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

编译通过，可以下载运行

单击 Go，全速运行，看到 LED1 点亮

点击 break 暂停运行，可以看到程序停在 while(1)处

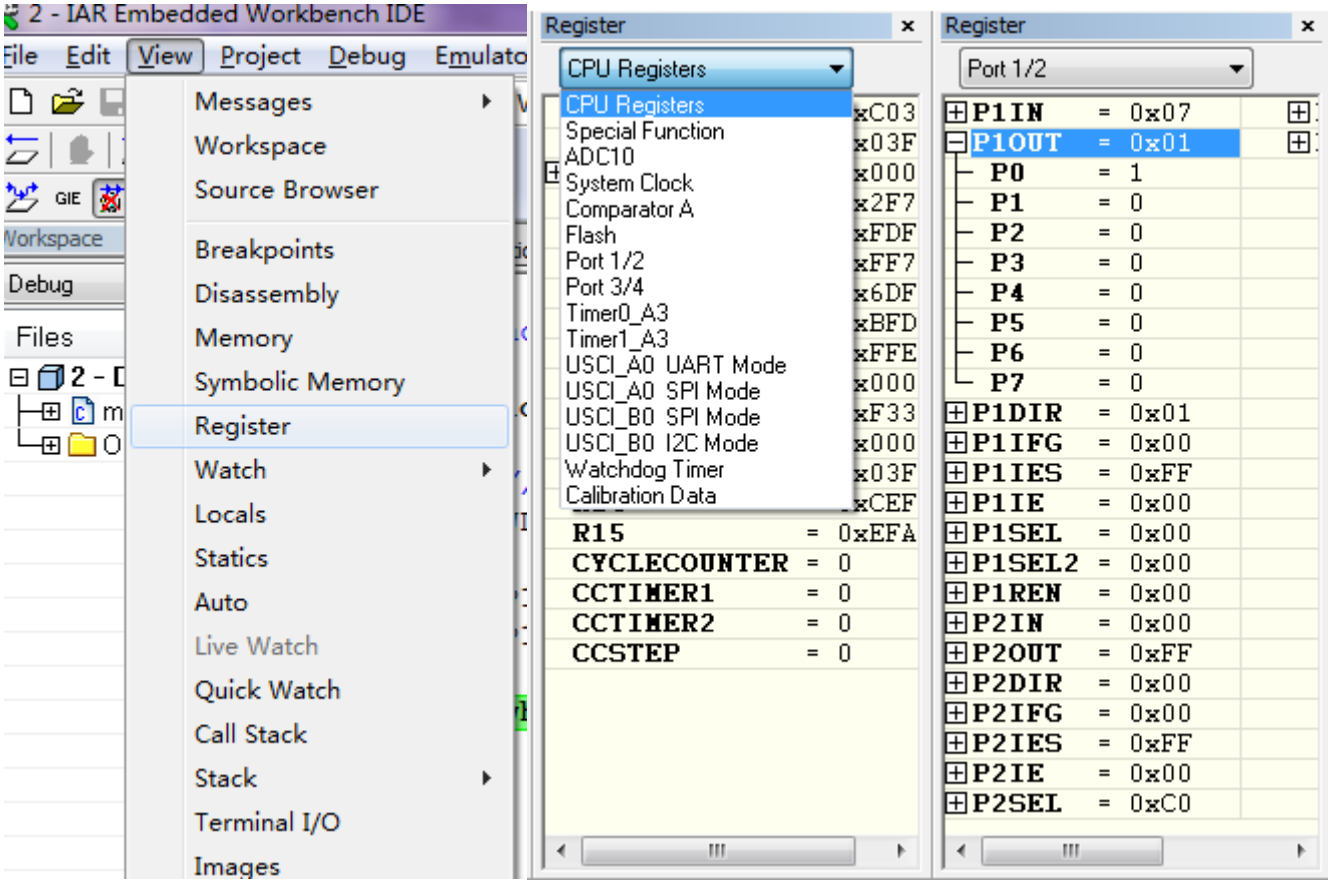
```
void main( void )
{
  // Stop watchdog timer to preven
  WDTCTL = WDTPW + WDTHOLD;

  P1DIR = BIT0;//P1.0设置为输出
  P1OUT = BIT0;//P1.0输出高电平

  while(1);

}
```

这时，我们打开 register 窗口，观察单片机内寄存器



可以看到 P1OUT 的值已经被我们改变，P1.0 为 1，其他为 0

五、LED 闪烁

顾名思义，LED 闪烁自然是 LED 等交替亮灭。但是，如果直接对控制 LED 的 IO 口循环取反，则会因为闪烁过快，看不出闪烁效果。为此，要加入延时程序，使其每隔一段时间取反一次。

延时一般有两种实现方式，空循环或定时延时。空循环使用 for，while 实现，简单但延时时间不准确；定时延时要使用定时器，时延准确但实现较为复杂。简单起见，我们使用空循环。

```
void delay(int i)
{
  while(i--);
}
```

这样即是最简单的延时函数，通过参数大小控制时延时间。

再修改 main 函数

```
void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  P1DIR = BIT0;//P1.0设置为输出
  P1OUT = BIT0;//P1.0输出高电平

  while(1)
  {
    P1OUT ^= BIT0;
    delay(20000);
  }

}
```

这样，LED 闪烁的程序就完成了。这里要注意延时数值不能超过 32768（int 的范围）。