

第3章 单片机程序结构和设计

第1节 单片机 C语言程序结构

第2节 单片机 C语言

端口寄存器定义及编程控制

第3节 单片机 C语言程序设计举例

微机控制台方式下用C语言编写的“Hello, World!”

```
#include <stdio.h>  
  
int main( )  
{  
  
    printf(“Hello, World !\n”);  
  
    return 0;  
  
}
```

通过调用系统提供的标准库函数实现屏幕输出

CCS下 C语言上机过程

一. 创建一个C语言项目

(Creat a New Project/

Empty project(with Main))

含项目相关属性设置

(MCU类型、项目类型等)

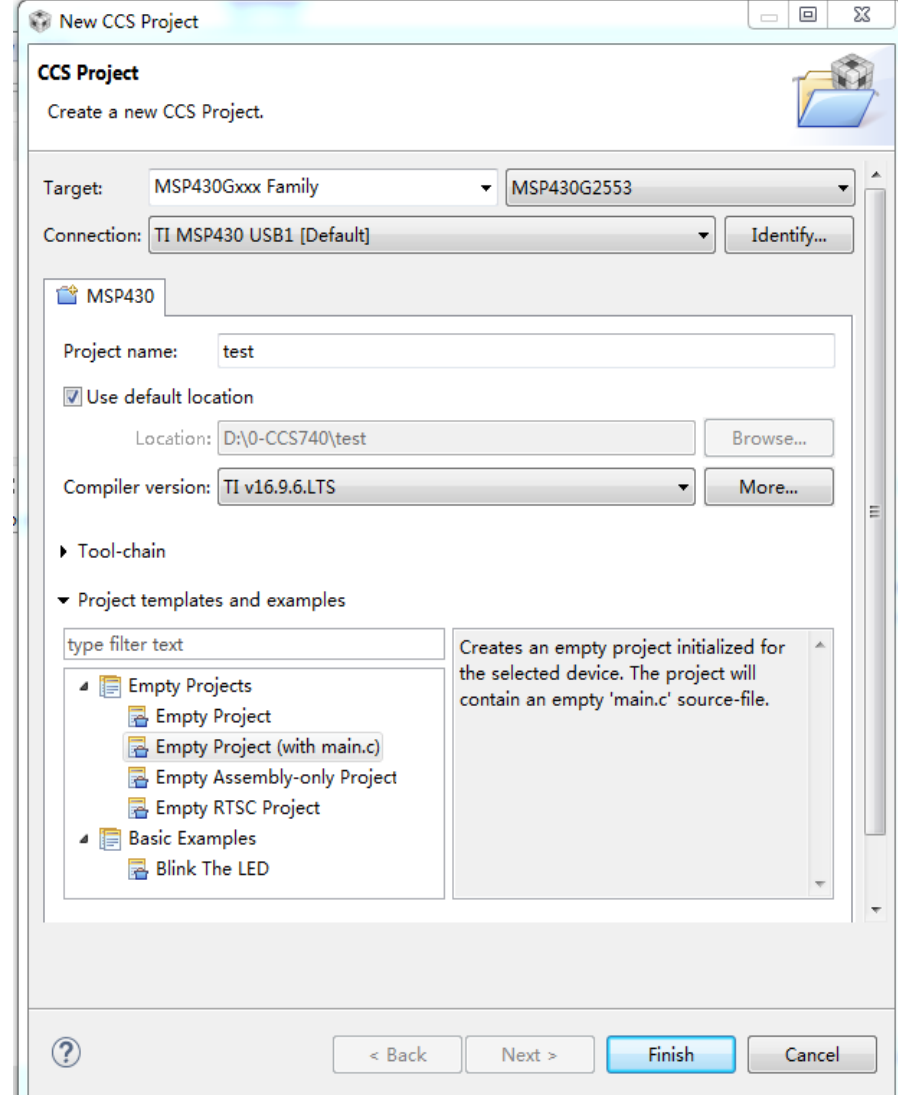
二. 编写源程序并添加到项目中

三. 编译和连接(Build Project)

五. 下载程序到目标MCU(Debug)

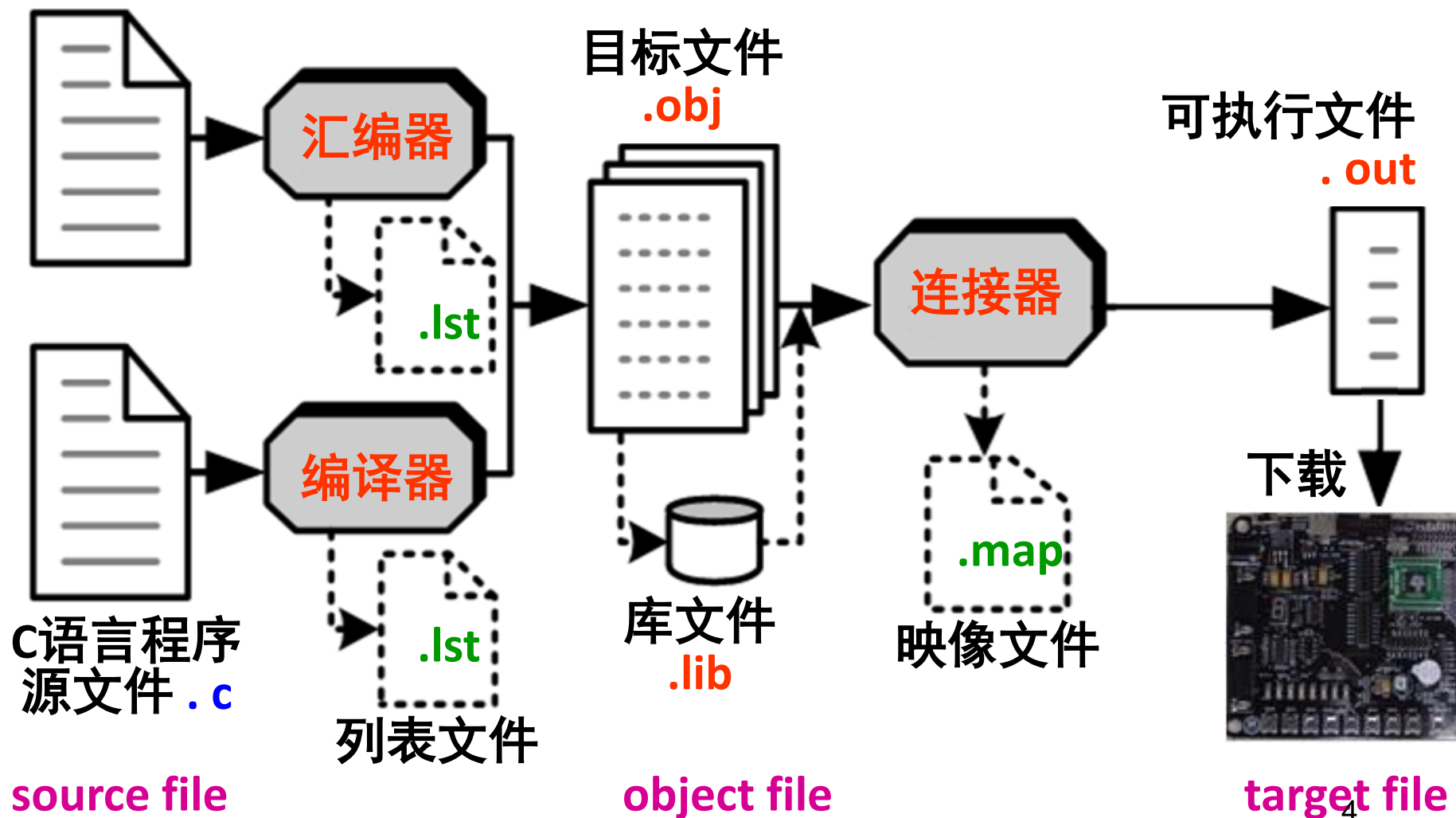
六. 调试和运行(Resume、Step、Step Over、View/Registers、memory

四、五、六中出现有问题， 返回二、三步骤



CCS 可执行文件生成过程

汇编语言程序
源文件 .asm



注意：关闭CCS编译器优化功能

设置编译器的优化级别。点击菜单栏Project > Properties，如图A-12在出现的窗口中，点击左侧的Build > MSP430 Compile > Optimization，将右侧的 Optimization level 从原来默认设置的0-Register Optimization 改置为 Off，即关闭优化功能。

建议本学期实验，无特别要求下，选off级调试。

第1节 单片机C语言程序结构

CCS下 msp430的C程序模板main.c

```
#include <msp430.h>
```

```
/**
```

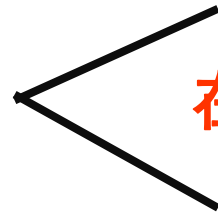
```
 * main.c
```

```
 */
```

```
int main(void)
```

```
{
```

```
    WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
```



在这里插入用户编写的程序部分

```
    return 0;
```

```
}
```

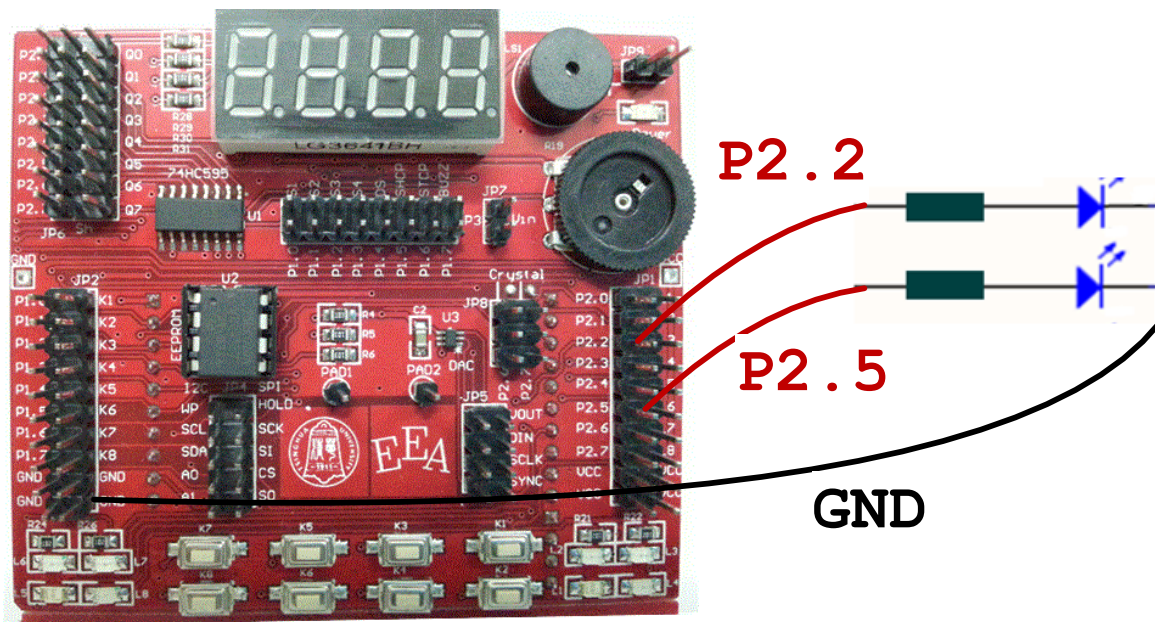
CCS的C语言程序结构与一般C程序基本相同

```
#include <msp430.h>           //包含头文件
//常量型存储器定义(在ROM区)
const char LEDtab[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
//带初始化全局变量定义(在RAM区)
unsigned char STD[45]="Hi , this is MSP430.";
//未初始化全局变量定义(在RAM区)
int  a, b, c;
//函数声明
void subN(形式参数);
//函数定义
int main ( void )             //用户程序入口函数
{
    数据说明部分;             //局部变量定义
    执行语句部分;
    subN();                   //函数调用
    .....
}
void subN (形式参数 )        //子函数
{
    数据说明部分;             //局部变量定义
    执行语句部分;
    .....
}
```

程序格式

- 程序由函数构成, 函数由语句组成
- 语句以分号“;”作为结束符, 注意不是分隔符
- 程序由**主函数main()**作为程序入口,
用户程序从函数main() 的第一条语句开始执行,
程序执行完毕的标志是函数main()中的代码执行完毕
- 标识符区分大小写,
不能在变量名、函数名、关键字中插入空格和空行
- **关键字及编译预处理命令用小写字母书写**
- 程序用大括号{ }表示程序的层次范围
- 程序没有行的概念, 可任意书写, 但为了可读性,
书写一对大括号时根据层次采用缩格和列向对齐方式。
- 注释部分用/*.....*/表示
或用“//”表示其后的内容为注释

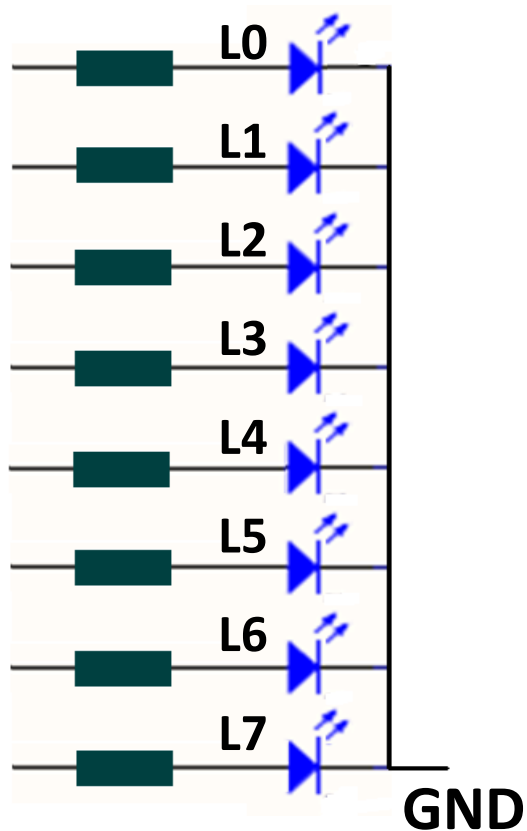
例：通过P2.2和P2.5管脚控制LED的闪烁
(可以将P2.2和P2.5用导线连接至实验板上的LED)



自己试验时可以接到实验板上的LED上，注意发光二极管的共阴或共阳接法

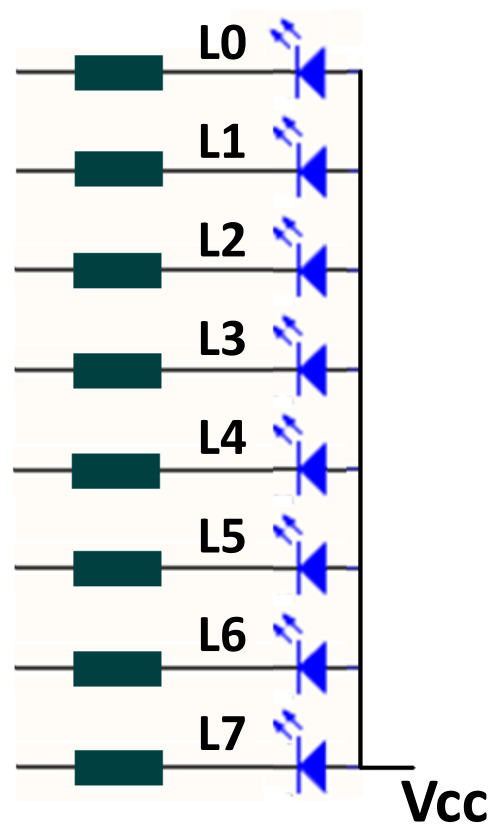
注意发光二极管的共阴、共阳接法

发光二极管电路图和工作原理



共阴连接

输出1时亮



共阳连接

输出0时亮

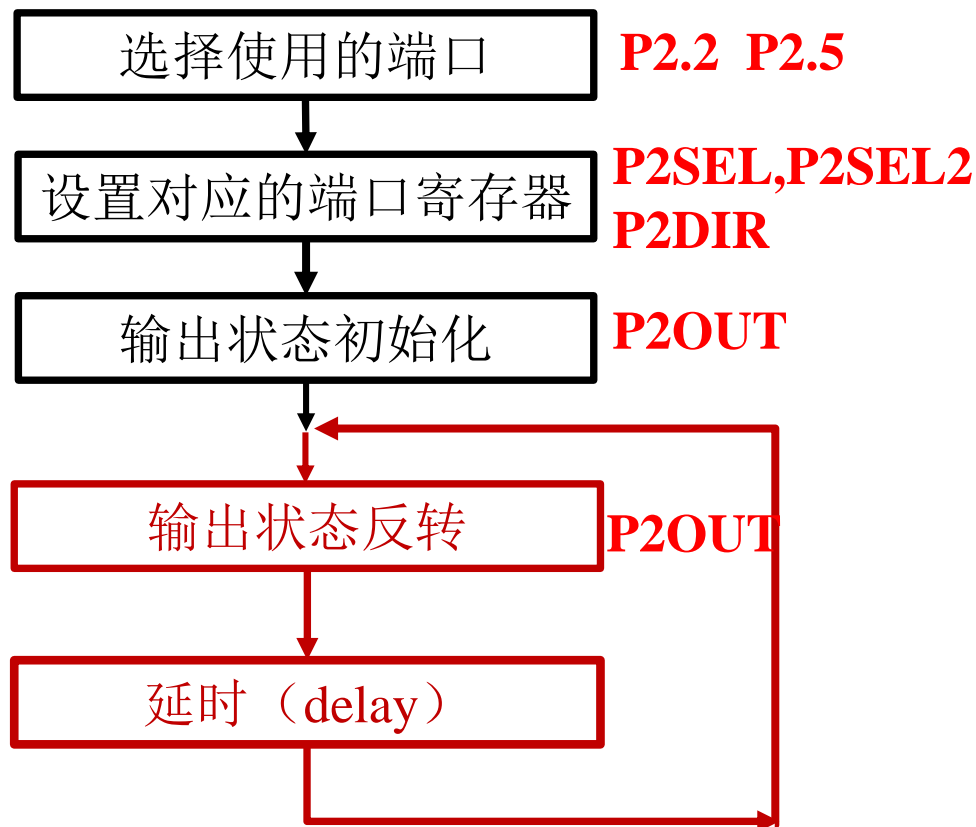
例：通过P2.2和P2.5管脚控制LED的闪烁

算法设计：

LED灯控制：通过对应的I/O端口管脚控制

如何闪烁：交替输出1和0

闪烁时间：延时一定时间后再变化



主程序含两部分：**初始化**部分和**主循环**部分；

初始化部分是对MCU的软、硬件相关部分做初始设置，

不同处：**主循环**部分是MCU实时处理部分，通常是一个无限循环。

test_g2553.c (主循环采用**for(;;)**)

```
#include "msp430.h"
int main ( void )
{
    unsigned int i;                //定义延时变量
    WDTCTL = WDTPW + WDT HOLD;    //关闭看门狗
    P2SEL &=~(BIT2+BIT5);         //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;            //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;            //设置端口P2.2和P2.5为输出方向
    for (;;)                      //主循环
    {
        P2OUT ^=(BIT2+BIT5);      //将P2.2和P2.5的值取反后输出
        for (i=0;i<0xffff;i++);  //延时
    };
}
```

主程序含两部分：**初始化**部分和**主循环**部分；

初始化部分是对MCU的软、硬件相关部分做初始设置，

不同处：**主循环**部分是MCU实时处理部分，通常是一个无限循环。

test_g2553.c (主循环采用**while(1)**)

```
#include "msp430.h"
int main ( void )
{
    unsigned int i;                //定义延时变量
    WDTCTL = WDTPW + WDT HOLD;    //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;            //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;            //设置端口P2.2和P2.5为输出方向
    while (1)                      //主循环
    {
        P2OUT ^=(BIT2+BIT5);      //将P2.2和P2.5的值取反后输出
        for (i=0;i<0xffff;i++);  //延时
    };
}
```

子函数的声明、定义和调用

test_g2553.c (采用while(1)和延时函数)

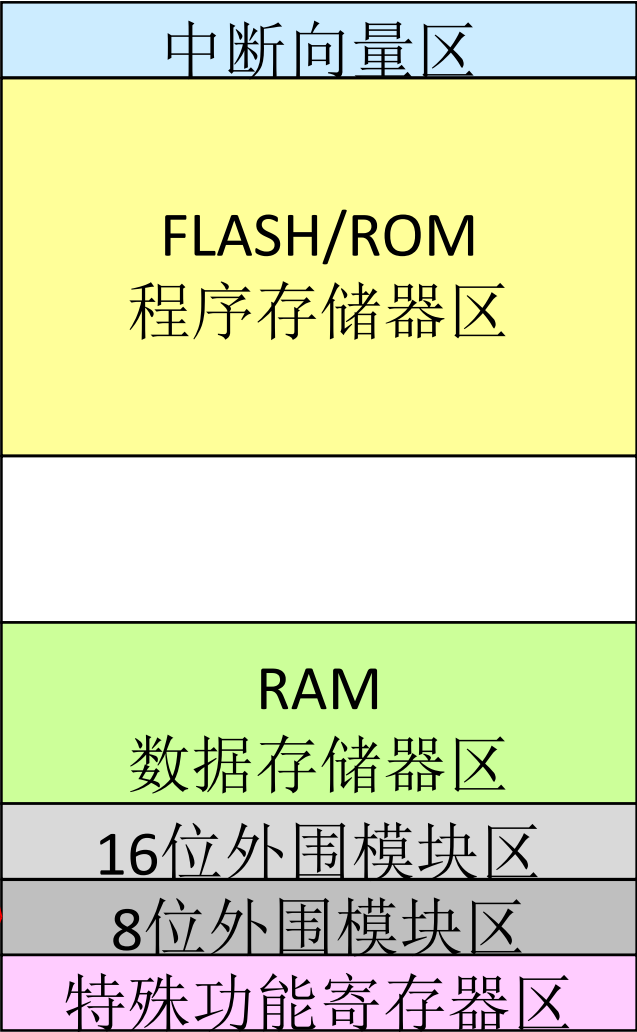
```
#include "msp430.h"
void delay( ) ;
int main ( void )
{
    unsigned int i;          //定义延时变量
    WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;        //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;        //设置端口P2.2和P2.5为输出方向
    while (1)                //主循环
    {
        P2OUT ^=(BIT2+BIT5); //将P2.2和P2.5的值取反后输出
        delay( ) ;           //延时
    };
}

void delay( )
{
    unsigned int i ;          //定义函数变量
    for ( i=0; i<0xffff; i++ ) ; //延时
}
```

CPU外部空间统一编址

地址FFFFH

端口	SEL 功能	DIR 方向	OUT 输出	IN 输入
属性	可读写	可读写	可读写	只读
复位值	00H	00H	-	-
P1	026H	022H	021H	020H
P2	02EH	02AH	029H	028H
P3	01BH	01AH	019H	018H



地址0000H

```

#include "msp430.h"
void delay( ) ;
int main ( void )
{
    unsigned int i;          //定义延时变量
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;        //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;        //设置端口P2.2和P2.5为输出方向
    while (1)                 //主循环
    {
        P2OUT ^=(BIT2+BIT5); //将P2.2和P2.5的值取反后输出
        delay( ) ;           //延时
    };
}
void delay( )
{
    unsigned int i ;          //定义函数变量
    for ( i=0; i<0xffff; i++ ) ; //延时
}

```

- “P2SEL” “P2OUT” 等变量是在哪里定义的
- “P2SEL” “P2OUT” 等是怎么对应到端口寄存器的地址的？
- BIT2和BIT5是什么意思，哪里定义的？
- &=~， |=是什么操作？

第2节 单片机 C语言I/O寄存器定义及编程控制

一、I/O寄存器的定义

二、编程操作I/O寄存器

三、编程控制I/O引脚进行输入/输出

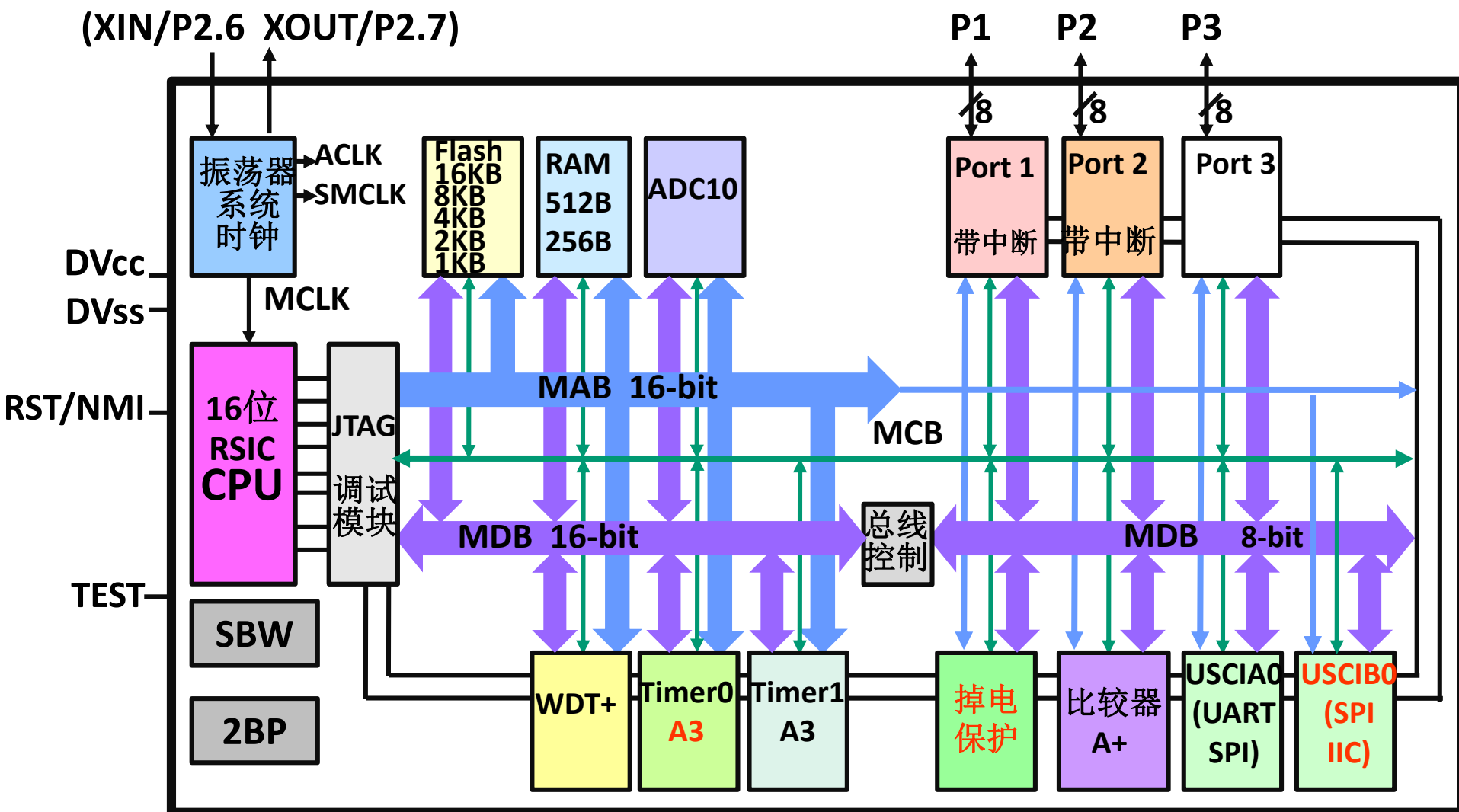
一、I/O寄存器的定义

- **I/O寄存器（端口寄存器）**：MCU内有丰富外围模块，这些模块内部包含CPU可以操作的I/O寄存器，又称I/O端口、端口寄存器

例如与基本输入/输出有关的端口寄存器

PxSEL, PxSEL2, PxDIR, PxOUT, PxIN, PxREN

MSP430G2553内部各模块含CPU可以操作的I/O寄存器



MSP430G2553内部各模块I/O寄存器例

模块	端口名称	端口地址
特殊功能	IE1	000h
	IE2	001h
	IFG1	002h
	IFG2	003h
P1	P1IN	020h
	P1OUT	021h
	P1DIR	022h
	P1IFG	023h
	P1IES	024h
	P1IE	025h
	P1SEL	026h
	P1REN	027h
	P1SEL2	041h

模块	端口名称	端口地址
P2	P2IN	028h
	P2OUT	029h
	P2DIR	02Ah
	P2IFG	02Bh
	P2IES	02Ch
	P2IE	02Dh
	P2SEL	02Eh
	P2REN	02Fh
	P2SEL2	042h
P3	P3IN	018h
	P3OUT	019h
	P3DIR	01Ah
	P3SEL	01Bh
	P3REN	010h
	P3SEL2	043h

模块	端口名称	端口地址
基本时钟	DCOCTL	056h
	BCSCTL1	057h
	BCSCTL2	058h
	BCSCTL3	053h
比较器A	CACTL1	059h
	CACTL2	05Ah
	CAPD	05Bh
ADC10	ADC10AE0	04Ah
	ADC10AE1	04Bh
	ADC10DT0	048h
	ADC10DT1	049h
	ADC10SA	1BCh
	ADC10MEM	1B4h
	ADC10CTL0	1B0h
	ADC10CTL1	1B2h

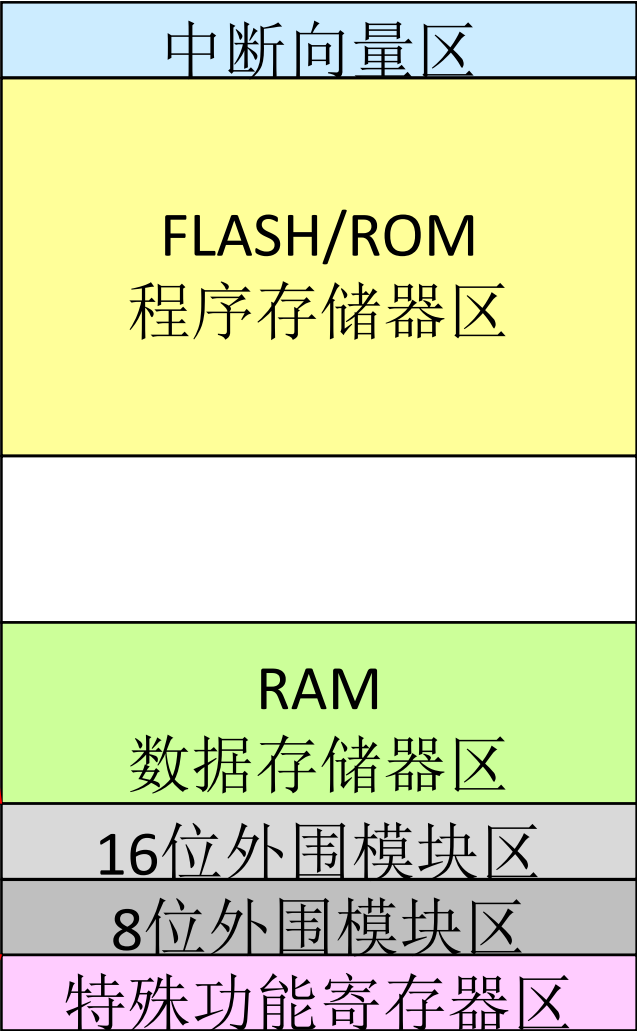
I/O寄存器

地址FFFFH

模块	端口名称	端口地址
特殊功能	IE1	000h
	IE2	001h
	IFG1	002h
	IFG2	003h
P1	P1IN	020h
	P1OUT	021h
	P1DIR	022h
	P1IFG	023h
	P1IES	024h
	P1IE	025h
	P1SEL	026h
	P1REN	027h
	P1SEL2	041h

模块	端口名称	端口地址
P2	P2IN	028h
	P2OUT	029h
	P2DIR	02Ah
	P2IFG	02Bh
	P2IES	02Ch
	P2IE	02Dh
	P2SEL	02Eh
	P2REN	02Fh
P3	P2SEL2	042h
	P3IN	018h
	P3OUT	019h
	P3DIR	01Ah
	P3SEL	01Bh
	P3REN	010h
	P3SEL2	043h

模块	端口名称	端口地址
基本时钟	DCOCTL	056h
	BCSCTL1	057h
	BCSCTL2	058h
	BCSCTL3	053h
比较器A	CACTL1	059h
	CACTL2	05Ah
	CAPD	05Bh
ADC10	ADC10AE0	04Ah
	ADC10AE1	04Bh
	ADC10DT0	048h
	ADC10DT1	049h
	ADC10SA	1BCh
	ADC10MEM	1B4h
	ADC10CTL0	1B0h
	ADC10CTL1	1B2h

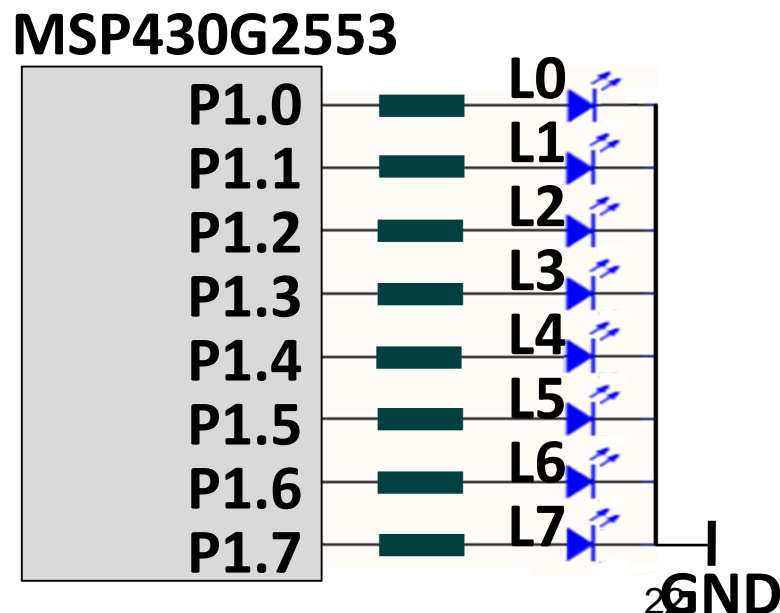


地址0000H

- CPU对外围模块的控制，是通过对模块内I/O寄存器的读/写操作完成。读操作获取其内容，写操作改变其内容。

例如 利用 端口1的8个引脚P1.7~P1.0控制外部发光二极管

需要设置 P1SEL、 P1SEL2为0x00， P2DIR为0xFF，
然后将P1OUT设置为P1.7~P1.0上所要的电平值



C语言程序中，如何编程让CPU操作这些I/O寄存器呢？

- 为方便用户使用单片机，集成开发平台(如TI的CCS或IAR 430), 根据各款单片机内部实际I/O寄存器的地址和功能，编写了与单片机底层硬件紧密相关的msp430xxxx.h等头文件，用符号的形式对各款单片机内部的I/O寄存器做了定义。用户可以用这些符号进行编程，完成对I/O寄存器的操作。

CCS下建立msp430的C项目时，自带的C程序模板

```
#include <msp430.h>

/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer

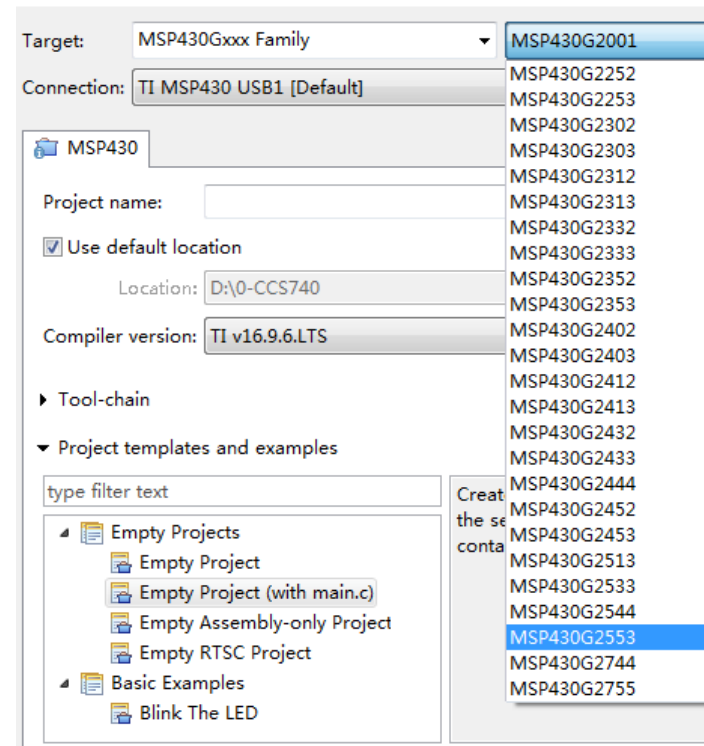
    return 0;
}
```

该程序中并未出现任何具体型号单片机的信息，
而且对所有选型的单片机，该模板程序都是一样的。
这样的模板程序有什么益处？

头文件 msp430.h

```
#if defined (__MSP430C111__)  
#include "msp430c111.h"  
  
.....  
  
#elif defined (__MSP430F149__)  
#include "msp430f149.h"  
  
.....  
  
#elif defined (__MSP430G2553__)  
#include "m430g2553.h "  
  
.....
```

利用 **msp430.h**，用户建立项目时选择的MCU型号，自动包含对应的**xxxx MCU**的 **msp430xxxx.h** 头文件。



例如：

选择单片机型号为 MSP430G2553，
相当于在系统定义了 **__MSP430G2553__**。

msp430.h头文件中 用 **#include "msp430g2533.h"**语句，
将 **msp430G2553.h** 头文件 包含到程序中。

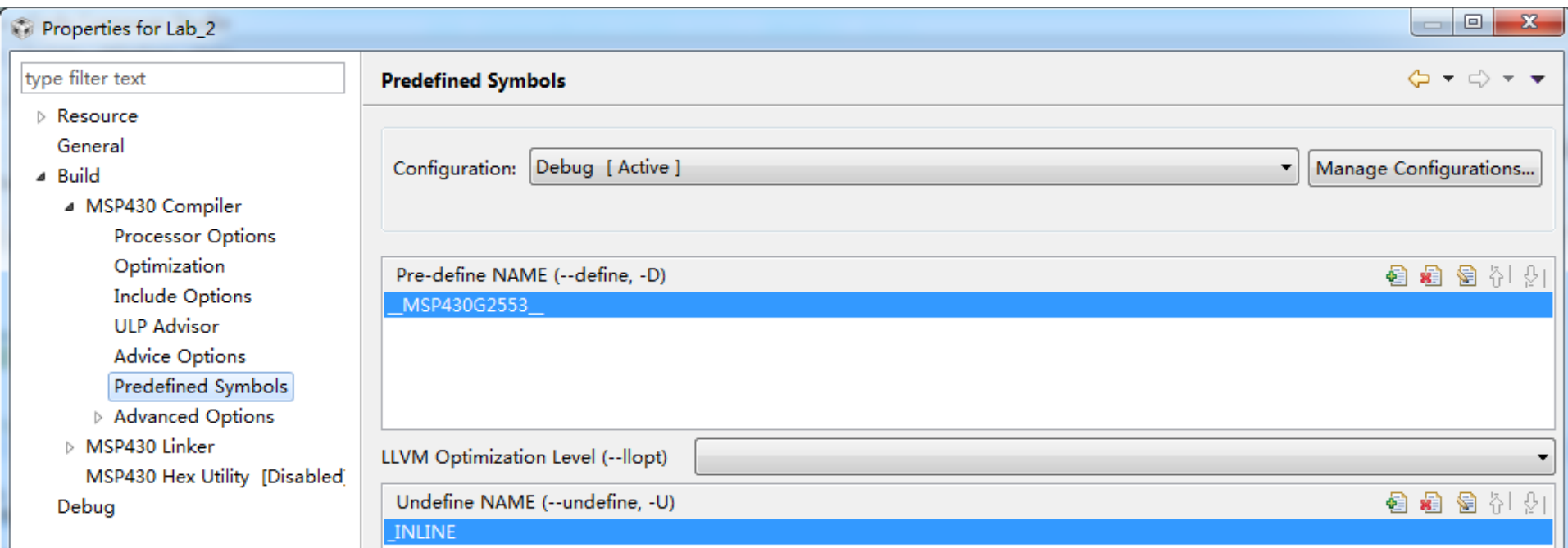
msp430.h

.....

```
#elif defined (__MSP430G2553__)  
#"include "msp430g2553.h" '26
```

查看预定义单片机类型符号

Project>Properties>Build>MSP430 Compiler>Predifined Symbles



CCS下建立msp430的C项目时，自带的C程序模板

```
#include <msp430.h>

/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer

    return 0;
}
```

由于源程序本身只有语句 **#include msp430.h**
即使**改选 msp430G2513** 单片机，
由CCS**利用msp430.h**文件找到对应的 **msp430G2513.h** 头文件，
而源程序本身不用改，使**程序的移植性比较好**

```
..... msp430.h
#elif defined (__MSP430G2513__)
#include "msp430g2513.h "
```

CCS下与msp430 C语言项目有关的三个关键文件

■ **msp430g2553.h**

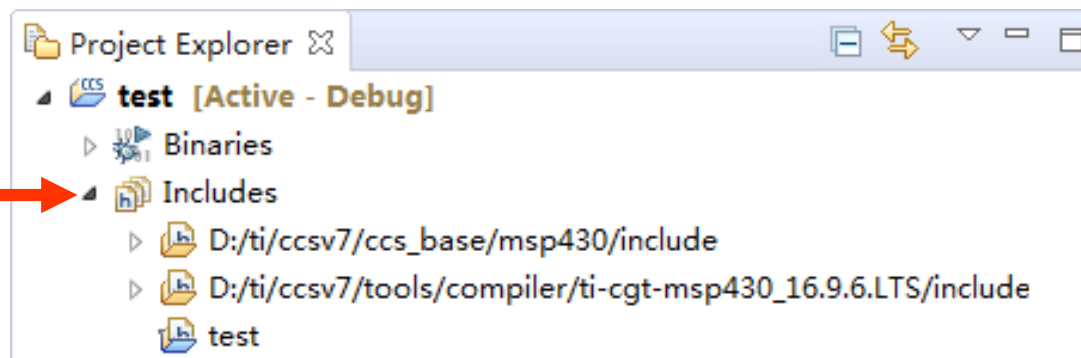
■ **msp430g2553.cmd**

■ **Lnk_msp430g2553.cmd**

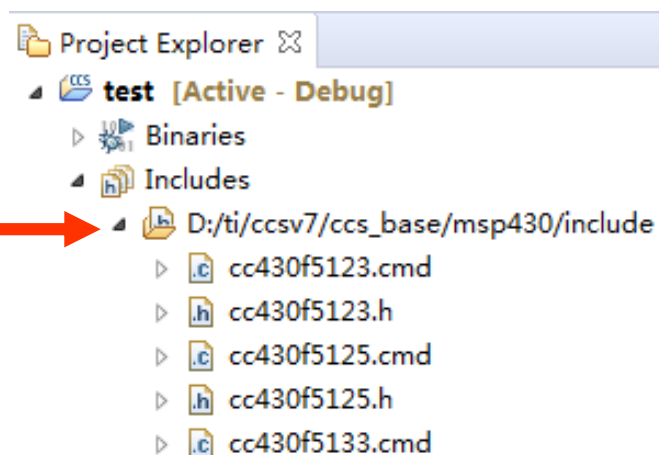
- 单片机硬件数据手册(datasheet)文字描述单片机的硬件信息；
- CCS开发软件通过上面三个文件将单片机的硬件信息“数字化”，编译、连接时利用这些文件，使软件与单片机底层硬件相对应。
- 是用户编程时的重要头文件。

查看相关文件的方法

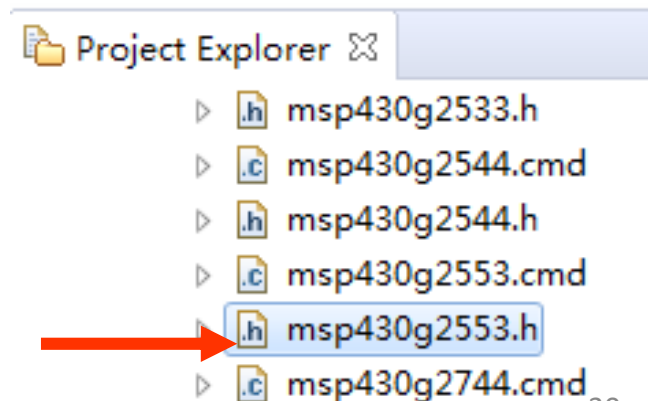
1) 点击项目中的includes



2) 点击项目中的include下的
.../ccs_base/msp430/include



3) 在列表中找到要查看的文件
msp430g2553.h
msp430g2553.cmd
Lnk_msp430g2553.cmd



```

#include "msp430.h"
void delay( ) ;
int main ( void )
{
    unsigned int i;          //定义延时变量
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;        //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;        //设置端口P2.2和P2.5为输出方向
    while (1)                 //主循环
    {
        P2OUT ^=(BIT2+BIT5); //将P2.2和P2.5的值取反后输出
        delay( ) ;           //延时
    };
}
void delay( )
{
    unsigned int i ;          //定义函数变量
    for ( i=0; i<0xffff; i++ ) ; //延时
}

```

- “P2SEL ” “P2OUT ” 等变量是在哪里定义的
- “P2SEL ” “P2OUT ” 等是怎么对应到端口寄存器的地址的？
- BIT2和BIT5是什么意思，哪里定义的？
- &=~， |=是什么操作？

■ msp430G2553.h

依据单片机底层寄存器硬件属性(char, int),
对单片机内寄存器以变量形式进行定义

字节属性

Table 15. 具有字节存取功能的外设

模块	寄存器说明	寄存器名称	偏移
端口 P1	端口 P1 选择 2	P1SEL2	041h
	端口 P1 电阻器使能	P1REN	027h
	端口 P1 选择	P1SEL	026h
	端口 P1 中断启用	P1IE	025h
	端口 P1 中断边沿选	P1IES	024h
	端口 P1 中断标志	P1IFG	023h
	端口 P1 方向	P1DIR	022h
	端口 P1 输出	P1OUT	021h
	端口 P1 输入	P1IN	020h

msp430g2553数据手册


```
#define SFR_8BIT(address)    extern volatile unsigned char address
```

```
SFR_8BIT(P1IN);           /* Port 1 Input */
SFR_8BIT(P1OUT);          /* Port 1 Output */
SFR_8BIT(P1DIR);          /* Port 1 Direction */
SFR_8BIT(P1IFG);          /* Port 1 Interrupt Flag */
SFR_8BIT(P1IES);          /* Port 1 Interrupt Edge Select */
SFR_8BIT(P1IE);           /* Port 1 Interrupt Enable */
SFR_8BIT(P1SEL);          /* Port 1 Selection */
SFR_8BIT(P1SEL2);         /* Port 1 Selection 2 */
SFR_8BIT(P1REN);          /* Port 1 Resistor Enable */
```



```
extern volatile unsigned char P1IN;
extern volatile unsigned char P1OUT;
extern volatile unsigned char P1DIR;
extern volatile unsigned char P1IFG;
extern volatile unsigned char P1IES;
extern volatile unsigned char P1IE;
extern volatile unsigned char P1SEL;
extern volatile unsigned char P1SEL2;
extern volatile unsigned char P1REN;
```

声明这些变量在外部定义了；
这些变量的类型是 char ；
Volatile 属性，说明是一个跟内存单元或IO寄存器单元有关的变量，告诉编译器对这种类型变量不要轻易做优化

#define SFR_16BIT(address) extern volatile unsigned int address

/* *****

字属性

* ADC10

*****:

Table 14. 具有字存取功能的外设

SFR_16BIT(ADC10CTL0); /* ADC10 Control 0 */
SFR_16BIT(ADC10CTL1); /* ADC10 Control 1 */
SFR_16BIT(ADC10MEM); /* ADC10 Memory */
SFR_16BIT(ADC10SA); /* ADC10 Data Transfer

模块	寄存器说明	寄存器名称	偏移
ADC10	ADC 数据传输起始地址	ADC10SA	1BCh
	ADC 内存	ADC10MEM	1B4h
	ADC 控制寄存器 1	ADC10CTL1	1B2h
	ADC 控制寄存器 0	ADC10CTL0	1B0h
Timer1_A3	捕获/比较寄存器	TA1CCR2	0196h
	捕获/比较寄存器	TA1CCR1	0194h
	捕获/比较寄存器	TA1CCR0	0192h
	Timer_A 寄存器	TA1R	0190h
	捕获/比较控制	TA1CCTL2	0186h
	捕获/比较控制	TA1CCTL1	0184h
	捕获/比较控制	TA1CCTL0	0182h
	Timer_A 控制	TA1CTL	0180h
	Timer_A 中断矢量	TA1IV	011Eh

/* *****

* Timer0_A3

*****:

SFR_16BIT(TA1IV); /* Timer1_A3 Interrupt Vector */
SFR_16BIT(TA1CTL); /* Timer1_A3 Control */
SFR_16BIT(TA1CCTL0); /* Timer1_A3 Capture/Compare 0 Control */
SFR_16BIT(TA1CCTL1); /* Timer1_A3 Capture/Compare 1 Control */
SFR_16BIT(TA1CCTL2); /* Timer1_A3 Capture/Compare 2 Control */
SFR_16BIT(TA1R); /* Timer1_A3 Counter Register */
SFR_16BIT(TA1CCR0); /* Timer1_A3 Capture/Compare 0 */
SFR_16BIT(TA1CCR1); /* Timer1_A3 Capture/Compare 1 */
SFR_16BIT(TA1CCR2); /* Timer1_A3 Capture/Compare 2 */

extern volatile unsigned int ADC10CTL0

```

#include "msp430.h"
void delay( ) ;
int main ( void )
{
    unsigned int i;          //定义延时变量
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;        //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;        //设置端口P2.2和P2.5为输出方向
    while (1)                 //主循环
    {
        P2OUT ^=(BIT2+BIT5); //将P2.2和P2.5的值取反后输出
        delay( ) ;           //延时
    };
}
void delay( )
{
    unsigned int i ;          //定义函数变量
    for ( i=0; i<0xffff; i++ ) ; //延时
}

```

- “P2SEL” “P2OUT” 等变量是在哪里定义的
- “P2SEL” “P2OUT” 等是怎么对应到端口寄存器的地址的？
- BIT2和BIT5是什么意思，哪里定义的？
- &=~， |=是什么操作？

■ msp430G2553.cmd

对单片机内部的I/O寄存器用符号进行定义，
并设置指向单片机底层的实际硬件地址

```
msp430g2553.cmd
90 *****/
91 P1IN      = 0x0020;
92 P1OUT     = 0x0021;
93 P1DIR     = 0x0022;
94 P1IFG     = 0x0023;
95 P1IES     = 0x0024;
96 P1IE      = 0x0025;
97 P1SEL     = 0x0026;
98 P1SEL2    = 0x0041;
99 P1REN     = 0x0027;
100 P2IN      = 0x0028;
101 P2OUT     = 0x0029;
102 P2DIR     = 0x002A;
103 P2IFG     = 0x002B;
104 P2IES     = 0x002C;
105 P2IE      = 0x002D;
106 P2SEL     = 0x002E;
107 P2SEL2    = 0x0042;
108 P2REN     = 0x002F;
```

Table 15. 具有字节存取功能的外设

模块	寄存器说明	寄存器名称	偏移
端口 P1	端口 P1 选择 2	P1SEL2	041h
	端口 P1 电阻器使能	P1REN	027h
	端口 P1 选择	P1SEL	026h
	端口 P1 中断启用	P1IE	025h
	端口 P1 中断边沿选	P1IES	024h
	端口 P1 中断标志	P1IFG	023h
	端口 P1 方向	P1DIR	022h
	端口 P1 输出	P1OUT	021h
	端口 P1 输入	P1IN	020h

msp430g2553数据手册

msp430G2553.cmd中的定义例

```
/* DIGITAL I/O Port1/2 Pull up / Pull down Resistors*/
```

```
P1IN    = 0x0020;  
P1OUT   = 0x0021;  
P1DIR   = 0x0022;  
P1IFG   = 0x0023;  
P1IES   = 0x0024;  
P1IE    = 0x0025;  
P1SEL   = 0x0026;  
P1SEL2  = 0x0041;  
P1REN   = 0x0027;  
P2IN    = 0x0028;  
P2OUT   = 0x0029;  
P2DIR   = 0x002A;  
P2IFG   = 0x002B;  
P2IES   = 0x002C;  
P2IE    = 0x002D;  
P2SEL   = 0x002E;  
P2SEL2  = 0x0042;  
P2REN   = 0x002F;
```

声明这些变量在存储系统中的实际地址

FFFFh FFE0h FFDFh	中断向量表
	FLASH/ROM 程序存储器区
↕	
↕	
	RAM 数据存储器区
0200h 01FFh 0100h	16位外围模块区
00FFh 0010h	8位外围模块区
000Fh 0000h	特殊功能寄存器区

连接时用到msp430G2553.cmd文件

■ Lnk_msp430G2553.cmd

声明单片机底层硬件相关信息

连接程序将软件和底层硬件相关联的重要依据

```
msp430g2553.h msp430g2553.cmd lnk_msp430g2553.cmd ✕
50 /*****
51 /* Specify the system memory map */
52 /*****
53
54 MEMORY
55 {
56     SFR : origin = 0x0000, length = 0x0010
57     PERIPHERALS_8BIT : origin = 0x0010, length = 0x00F0
58     PERIPHERALS_16BIT : origin = 0x0100, length = 0x0100
59     RAM : origin = 0x0200, length = 0x0200
60     INFOA : origin = 0x10C0, length = 0x0040
61     INFOB : origin = 0x1080, length = 0x0040
62     INFOC : origin = 0x1040, length = 0x0040
63     INFOD : origin = 0x1000, length = 0x0040
64     FLASH : origin = 0xC000, length = 0x3FDE
65     BSLSIGNATURE : origin = 0xFFDE, length = 0x0002,
66     INT00 : origin = 0xFFE0, length = 0x0002
67     INT01 : origin = 0xFFE2, length = 0x0002
68     INT02 : origin = 0xFFE4, length = 0x0002
69     INT03 : origin = 0xFFE6, length = 0x0002
70     INT04 : origin = 0xFFE8, length = 0x0002
71     INT05 : origin = 0xFFEA, length = 0x0002
72     INT06 : origin = 0xFFEC, length = 0x0002
73     INT07 : origin = 0xFFEE, length = 0x0002
74     INT08 : origin = 0xFFFF, length = 0x0002
75     INT09 : origin = 0xFFFF, length = 0x0002
76     INT10 : origin = 0xFFFF, length = 0x0002
77     INT11 : origin = 0xFFFF, length = 0x0002
78     INT12 : origin = 0xFFFF, length = 0x0002
79     INT13 : origin = 0xFFFF, length = 0x0002
80     INT14 : origin = 0xFFFF, length = 0x0002
81     RESET : origin = 0xFFFF, length = 0x0002
82 }
```

Table 8. 内存组织

		MSP430G2553 MSP430G2513
内存	尺寸	16kB
主：中断矢量	闪存	0xFFFF 至 0xFFC0
主：代码内存	闪存	0xFFFF 至 0xC000
信息内存	尺寸	256 字节
	闪存	010FFh 至 01000h
RAM	尺寸	512 字节
		0x03FF 至 0x0200
外设	16 位	01FFh 至 0100h
	8 位	0FFh 至 010h
	8 位 SFR	0Fh 至 00h

msp430g2553数据手册

Lnk_msp430G2553.cmd声明单片机底层硬件相关信息 连接程序将软件和底层硬件相关联的重要依据

```
/* **** */
/* Specify the system memory map */
/* **** */
MEMORY
{
    SFR : origin = 0x0000, length = 0x0010
    PERIPHERALS_8BIT : origin = 0x0010, length = 0x00F0
    PERIPHERALS_16BIT : origin = 0x0100, length = 0x0100
    RAM : origin = 0x0200, length = 0x0200
    FLASH : origin = 0xC000, length = 0x3FDE
    .....
}
```

二、编程操作端口寄存器

在msp430xxxx.h头文件中对端口寄存器做了定义后，对端口寄存器的操作与对变量的操作相似，直接使用端口寄存器的符号，就可操作端口寄存器

- 端口寄存器的写操作
- 端口寄存器的读操作


```

#include "msp430.h"
void delay( ) ;
int main ( void )
{
    unsigned int i;           //定义延时变量
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    P2SEL &=~(BIT2+BIT5); //设置引脚P2.2和P2.5为基本输入输出功能
    P2SEL2 &=~(BIT2+BIT5);
    P2OUT |=BIT2+BIT5;        //设置引脚P2.2和P2.5输出的初值为1
    P2DIR |=BIT2+BIT5;        //设置端口P2.2和P2.5为输出方向
    while (1)                 //主循环
    {
        P2OUT ^=(BIT2+BIT5); //将P2.2和P2.5的值取反后输出
        delay( ) ;           //延时
    };
}
void delay( )
{
    unsigned int i ;          //定义函数变量
    for ( i=0; i<0xffff; i++ ) ; //延时
}

```

- “P2SEL” “P2OUT” 等变量是在哪里定义的
- “P2SEL” “P2OUT” 等是怎么对应到端口寄存器的地址的？
- BIT2和BIT5是什么意思，哪里定义的？
- &=~， |=是什么操作？

■ 十进制与二进制（十六进制）

8

1	0	0	0
---	---	---	---

4

0	1	0	0
---	---	---	---

2

0	0	1	0
---	---	---	---

1

0	0	0	1
---	---	---	---

举例

3

0	0	1	1
---	---	---	---

5

0	1	0	1
---	---	---	---

11 (B)

1	0	1	1
---	---	---	---

15 (F)

1	1	1	1
---	---	---	---

■ 十进制与二进制（十六进制）

举例

0x13

0	0	0	1
---	---	---	---

0	0	1	1
---	---	---	---

0x55

0	1	0	1
---	---	---	---

0	1	0	1
---	---	---	---

0xFB

1	1	1	1
---	---	---	---

1	0	1	1
---	---	---	---

0xFF

1	1	1	1
---	---	---	---

1	1	1	1
---	---	---	---

■ 端口寄存器的写操作

改变端口寄存器的内容，就会包含对端口的写操作。

使用“=”等赋值操作，可以完成对端口的写操作，

写操作时，I/O端口在赋值号“=”的左边

■ 端口寄存器的写操作，例如：

```
P1SEL=0x00;           //置P1SEL为0
P1SEL2=0x00;          //置P1SEL2为0
P1OUT=0x55;           //置P1OUT为0x55
P1DIR=0xFF;           //设置P1DIR为0xFF
P1REN=0xFF;           //设置P1REN为0xFF
```

■ 端口寄存器的读操作

获取端口寄存器的内容，但不改变其内容的操作，都会包含对端口的读操作。

读操作时，I/O端口在赋值号“=”的右边表达式中。

■ 端口寄存器的读操作，例如：

```
unsigned char data, key;           //变量定义
data = P1IN;    //读取端口寄存器P1IN的内容赋值给变量
key = P2IN;     //读取端口寄存器P2IN的内容赋值给变量
```

- 赋值语句是对整个字节进行写操作，所有位的值都被赋值。

0x55

0	1	0	1
---	---	---	---


0	1	0	1
---	---	---	---

当前P1OUT状态为0x55

如果只希望将P1.0置为零，该怎么处理？

0	1	0	1
---	---	---	---

0	1	0	0
---	---	---	---



0x54

P1OUT = 0x54;

- 赋值语句是对整个字节进行写操作，所有位的值都被赋值。
- 如果只想改变其中的某些位，可先将原端口字节的内容读出来，利用与、或等逻辑操作，实现对某位、某些位的写操作，而其他位不受影响。

例 将P1.0 置1、 P1.0 置0，不影响其他位：

P1OUT |= 0x01; // “按位或”，相当于置1

P1OUT &= ~0x01; // 取反后再“按位与”，相当于置0

比较下面2条语句与上面2条语句的不同：

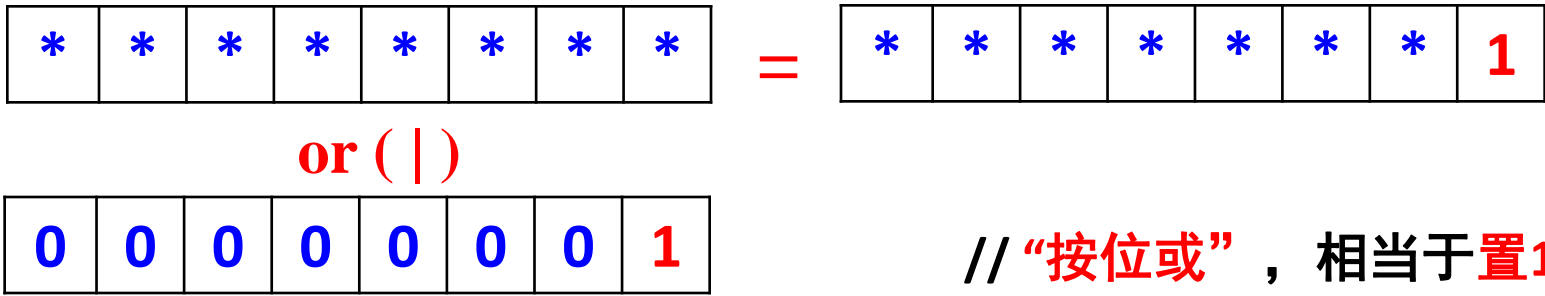
P1OUT = 0x01;

P1OUT = ~0x01;

如何值改变某一位的输出状态?

赋值操作	逻辑操作	
P1OUT = 0x01;	P1OUT = 0x01;	// “按位或”，相当于置1
P1OUT = ~0x01;	P1OUT &= ~0x01;	// 取反后再“按位与”，相当于置0

对于P1端口有何影响?



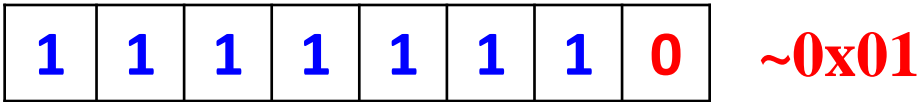
如何值改变某一位的输出状态?

赋值操作	逻辑操作	
P1OUT = 0x01;	P1OUT = 0x01;	// “按位或”，相当于置1
P1OUT = ~0x01;	P1OUT &= ~0x01;	// 取反后再“按位与”，相当于置0

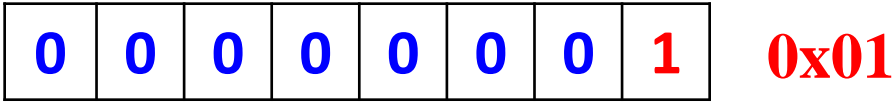
对于P1端口有何影响?



and (&)



// 取反后再“按位与”，相当于置0



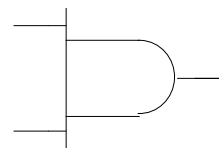
- 赋值语句是对整个字节进行写操作，所有位的值都被赋值。
- 如果只想改变其中的某些位，可利用与、或等逻辑操作，实现对某位、某些位的写操作，而其他位不受影响。

逻辑运算

1. “与” 运算 (and, &)

两位同时为1时结果为1，否则为0

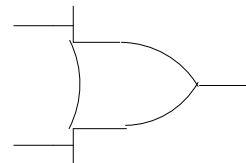
$0\&0=0$ $0\&1=0$ $1\&1=1$



2. “或” 运算 (or, |)

两位同时为0时结果为0，只要有1位是1结果就为1

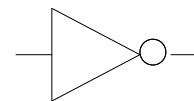
$0|0=0$ $0|1=1$ $1|1=1$



3. “非” 运算 (not, ~)

取反

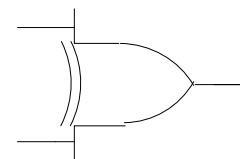
$\sim 1=0$ $\sim 0=1$



4. “异或” 运算 (xor, ^)

两位相同结果为0，两位不同结果为1

$1\wedge 1=0$ $0\wedge 0=0$ $1\wedge 0=1$



逻辑运算的方法（按位改变）

■ 使某位为1（“或”操作）

用该位和“1”相“或”，不变的位和“0”相“或”

`data |= 0x08;` //置data的D3位为1, 0x08=0000**1**000B

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	1	*	*	*

or (|)

0	0	0	0	1	0	0	0
---	---	---	---	----------	---	---	---

0x08

`data |= 0x10;` //置data的D4位为1, 0x10=000**1**0000B

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	1	*	*	*	*

or (|)

0	0	0	1	0	0	0	0
---	---	---	----------	---	---	---	---

0x10

`data |= 0x18;` //置data的D3和D4两位为1, 0x18=000**11**000B

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	1	1	*	*	*

or (|)

0	0	0	1	1	0	0	0
---	---	---	----------	----------	---	---	---

0x18

逻辑运算的方法（按位改变） 使用位值符号

■ 使某位为0（“与”操作）

用该位同“0”相“与”，不变的位同“1”相“与”

`data &= ~0x08;` //置data的D3位为0

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	0	*	*	*

and (&)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	0	1	1	1

 ~0x08

`data &= ~0x10;` //置data的D4位为0

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	0	*	*	*	*

and (&)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	1	1	1	1

 ~0x10

`data &= ~0x18;` //置data的D3和D4两位为0

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	*

 =

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	0	0	*	*	*

and (&)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	1	1

 ~0x18

逻辑运算的方法（按位改变）

使用位值符号

- 使某位求反（“异或”操作）
用该位同“1”异或，不变的位同“0”异或。

data ^= 0x08; //对data的D3位求反

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	*	1/0	*	*	*
---	---	---	---	-----	---	---	---

=

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	*	0/1	*	*	*
---	---	---	---	-----	---	---	---

xor (^)

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

0x08

data ^= 0x10; //对data的D4位求反

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	1/0	*	*	*	*
---	---	---	-----	---	---	---	---

=

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	0/1	*	*	*	*
---	---	---	-----	---	---	---	---

xor (^)

0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

0x10

data ^= 0x18; //对data的D3和D4两位求反

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	1/0	1/0	*	*	*
---	---	---	-----	-----	---	---	---

=

D7 D6 D5 D4 D3 D2 D1 D0

*	*	*	0/1	0/1	*	*	*
---	---	---	-----	-----	---	---	---

xor (^)

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

0x18

msp430G2553.h(续)

同时为提高编程的可读性，
并用符号对寄存器中的相关位进行定义

 msp430g2553.h

```
90 /*****
91 * STATUS REGISTER BITS
92 *****/
93
94 #define C      (0x0001)
95 #define Z      (0x0002)
96 #define N      (0x0004)
97 #define V      (0x0100)
98 #define GIE    (0x0008)
99 #define CPUOFF (0x0010)
100 #define OSCOFF (0x0020)
101 #define SCG0   (0x0040)
102 #define SCG1   (0x0080)
```

 msp430g2553.h

```
69 /*****
70 * STANDARD BITS
71 *****/
72
73 #define BIT0    (0x0001)
74 #define BIT1    (0x0002)
75 #define BIT2    (0x0004)
76 #define BIT3    (0x0008)
77 #define BIT4    (0x0010)
78 #define BIT5    (0x0020)
79 #define BIT6    (0x0040)
80 #define BIT7    (0x0080)
81 #define BIT8    (0x0100)
82 #define BIT9    (0x0200)
83 #define BITA    (0x0400)
84 #define BITB    (0x0800)
85 #define BITC    (0x1000)
86 #define BITD    (0x2000)
87 #define BITE    (0x4000)
88 #define BITF    (0x8000)
```

CPU的SR状态寄存器

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCOff	CPUoff	GIE	N	Z	C

mcp430G2553.h文件

对字中各位的值用符号定义，
使只有指定的位为1，其他位均为0，方便编程

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
											
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0020
											
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000

使用符号定义的位值，可提高程序的可读性。

```
#define BIT0 (0x0001)
#define BIT1 (0x0002)
#define BIT2 (0x0004)
#define BIT3 (0x0008)
#define BIT4 (0x0010)
#define BIT5 (0x0020)
#define BIT6 (0x0040)
#define BIT7 (0x0080)
#define BIT8 (0x0100)
#define BIT9 (0x0200)
#define BITA (0x0400)
#define BITB (0x0800)
#define BITC (0x1000)
#define BITD (0x2000)
#define BITE (0x4000)
#define BITF (0x8000)
```

逻辑运算的方法（按位改变） 使操作数的某位为0或1

■ 使某位为1（“或”操作）

用该位和“1”相“或”，不变的位和“0”相“或”

`data |= BIT3;` //置data的D3位为1, 0x08=0000**1**000B

`data |= BIT4;` //置data的D4位为1, 0x10=000**1**0000B

`data |= BIT3+BIT4;` //置data的D3和D4两位为1, 0x18=000**11**000B

■ 使某位为0（“与”操作）

用该位同“0”相“与”，不变的位同“1”相“与”

`data &= ~BIT3;` //置data的D3位为0

`data &= ~BIT4;` //置data的D4位为0

`data &= ~(BIT3+BIT4);` //置data的D3和D4两位为0

■ 使某位求反（“异或”操作）

用该位同“1”异或，不变的位同“0”异或。

`data ^= BIT3;` //对data的D3位求反

`data ^= BIT4;` //对data的D4位求反

`data ^= BIT3+BIT4;` //对data的D3和D4两位求反

■ 根据相与的结果，可以测试某位的值

例 测试data中的D1位是否为0?

```
if ( (data & BIT1) == 0 ) { .....};
```

例 测试data中的D4位是否为1?

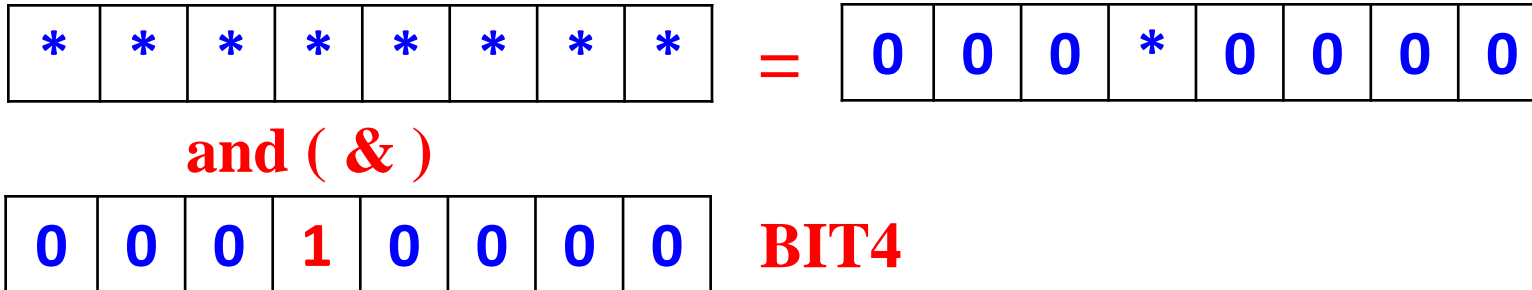
```
if ( (data & BIT4) != 0 ) { .....};
```

在if的条件判断语句中，包含了对端口的读操作。

思考： // 测试data中的D4位是否为1 ?
可否用 if ((data & BIT4) == 1) {.....};
为什么?

例 测试data中的D4位是否为1?

```
if ( (data & BIT4) != 0 ) { .....};
```



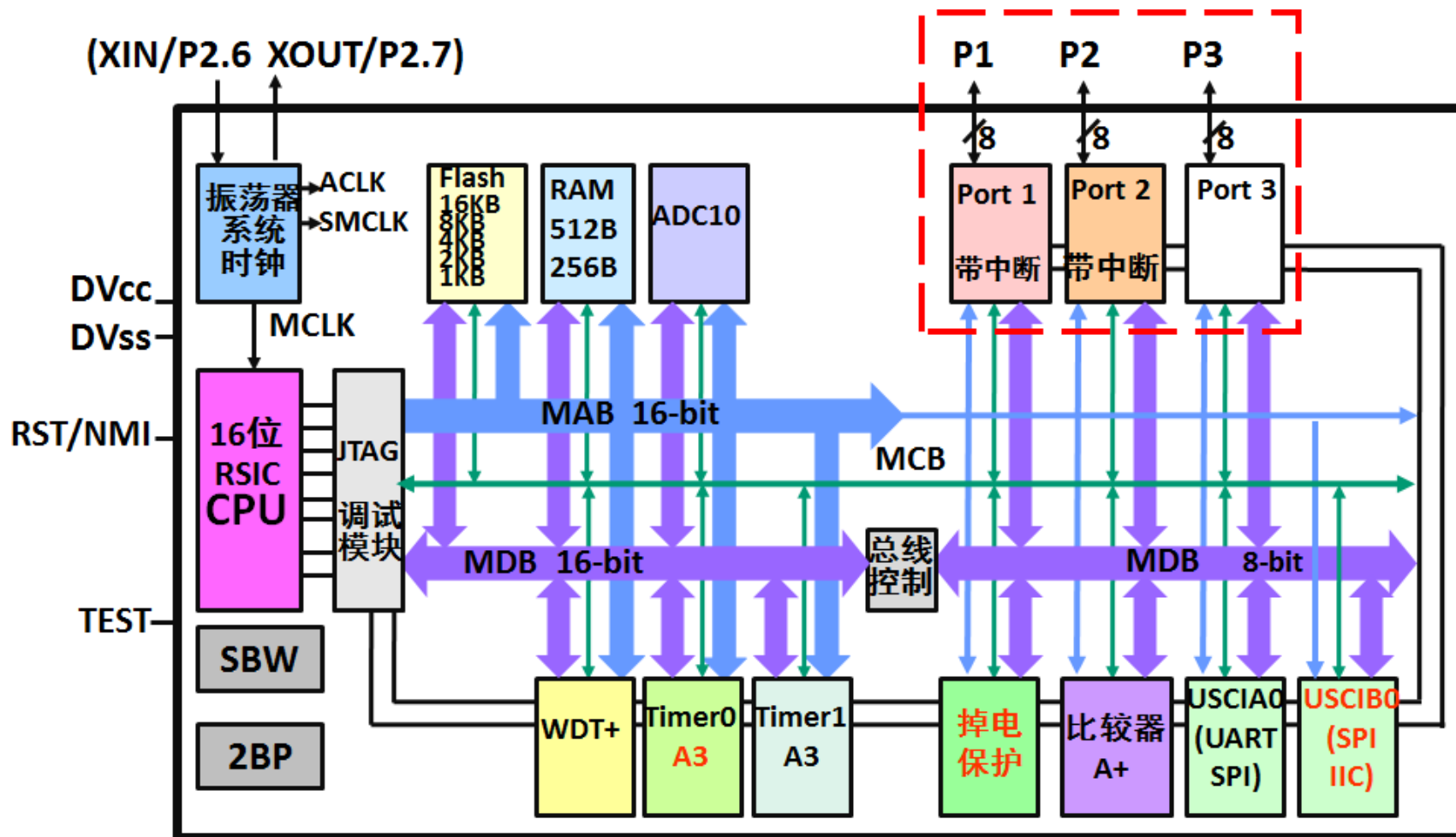
// 测试data中的D4位是否为1 ?

可否用 `if ((data & BIT4) == 1) {.....};`

为什么?

注意： CCS对只读属性的IO寄存器，
如P1IN，以及一些IO状态寄存器，
并没有特殊的声明和定义，
`P2IN=2; //`
这样的语句编译器并不报错，
但因底层硬件上，P2IN不可写，
所以执行的结果是 P2IN的值不受此句影响

三、编程控制I/O引脚进行输入/输出



28引脚、32引脚的MSP430G2x53有24个i/o引脚
20引脚的MSP430G2x53有16个i/o引脚

1. 利用端口Px的引脚进行输入/输出编程方法 (8个引脚)

1) 置PxSEL为0、 PxSEL2为0，将Px端口设为基本I/O功能

例 **P1SEL=0x00;** //设置P1的8个引脚均为基本I/O
P1SEL2=0x00; //

2) 设置PxDIR，确定端口方向

➤ 置PxDIR为0，则端口Px为输入端口方向

例 **P1DIR=0x00;** //设置P1的8个引脚均为输入

➤ 置PxDIR为0xff，则端口Px为输出端口方向

例 **P1DIR=0xff;** //设置P1的8个引脚均为输出

利用端口Px引脚进行输入/输出编程方法 (8个引脚)(续)

- 3) 作为输入的端口，如需要使用内部的上下拉电阻，需在初始化时，设置PxREN、PxOUT寄存器

例 设置 P1.7~P1.0 使用内部上拉电阻

P1OUT=0xff; //接到高电平成上拉电阻

P1REN=0xFF; //使能内部电阻

- 4) 作为输出的端口，向PxOUT赋值，可将信号输出到Px引脚上

例 **P1OUT=tempout;** //将变量tempout的内容通过端口P1输出

- 5) 读取 PxIN的内容，可获得端口Px引脚的电平

例 **tempin=P2IN;** //读入端口P2电平到tempin变量中

2. 对某Px.y引脚进行输入/输出编程方法(某个/某几个引脚)

1) 置PxSEL、 PxSEL2的y位为0，使引脚Px.y为基本I/O功能

例 **P1SEL &= ~BIT1;** //设置P1.1引脚为基本I/O
 P1SEL2 &= ~BIT1; //

2) 设置PxDIR 的y位，确定引脚Px.y方向

➤ 置PxDIR的y位为0，则引脚Px.y为输入方向

例 **P1DIR &= ~BIT1;** //设置引脚P1.1为输入方向

➤ 置PxDIR的y位为1，则引脚Px.y为输出方向

例 **P1DIR |= BIT1;** //设置引脚P1.1为输出方向

对某Px.y引脚进行输入/输出编程方法(某个/某几个引脚)(续)

- 3) 作为输入的Px.y引脚, 如需要使用内部的上下拉电阻, 需在初始化时设置PxREN、PxOUT寄存器的y位

例 设置P1.1使用内部上拉电阻

P1OUT|=BIT1; //接到高电平成上拉电阻

P1REN|=BIT1; //使能内部电阻

- 4) 向PxOUT的y位赋值, 可将信号输出到Px.y引脚上

例 **P1OUT &= ~BIT1;** //引脚P1.1输出低电平

P1OUT |= BIT1; //引脚P1.1输出高电平

- 5) 读取 PxIN.y的内容, 可获得端口Px.y 引脚的电平 (下节课介绍)

例 **tempin = P1IN & BIT1;** //读入引脚P1.1的电平到tempin变量中

if ((P1IN&BIT1)!=0) { }; //判断P1.1的输入是否高电平

if ((P1IN&BIT1)==0) { }; //判断P1.1的输入是否低电平