MSP430F149操作指南

**作者:向仔州**

**IAR EW430 5.3版本IDE安装...........................................................................................................2**

**BSL软件下载txt程序进单片机....................................................................................................7**

**BSL下载程序是通过串口下载，一定要安装x64版本的CH340,或者CH341芯片驱动。………..8**

**IAR 无法同时打开两个工程…………………………………………………………………………………..9**

**时钟系统……………………………………………………………………………………………………………..10**

**外部8M晶振启动，LED闪烁实验………………………………………………………………………...11**

**IAR自定义多目录，多C/H文件路径加载方法...................................................................................12**

**为了方便，可以使用$PROJ\_DIR$相对路径的方式来加载h文件，这样换个电脑也可以编译，不用重新修改路径…………………………………………………………………………………………………………..14**

**IAR 设置MSP430单片机堆栈大小…………………………………………………………………………....15**

**查看程序每个函数的栈使用大小.........................................................................................................16**

**GPIO操作.........................................................................................................................................17**

**MSP430 JTAG仿真器使用...............................................................................................................19**

**编译警告pointless integer comparison, the result is always false....................................21**

**GPIO外部中断实验...........................................................................................................................22**

**串口通信………………………………………………………………………………………………………….....23**

**串口printf实现.........................................................................................................................24**

**串口0接受中断实现………………………………………………………………………………………...25**

**16位定时器A使用...........................................................................................................................26**

**引脚输入信号电平捕获(定时器输入捕获)………………………………………………………………...31**

**ADC使用…………………………………………………………………………………………………….……..36**

**单次ADC转换，使用P6.0/A0通道进行测试………………………………………………………….38**

**IAR经过TI JTAG仿真器下载，如果某一天出现Fatal error: Failed to initialize. Check if hardware is connected. Check if drivers are installed. Try to restart the computer. Tools using the parallel port are not supported on Windows Vista Session aborted!...............................41**

**MSP430 因为是16位单片机 函数返回值必须满足16位。如果返回值是32位数据类型，就会有问题 (重点)...............................................................................................................................................42**

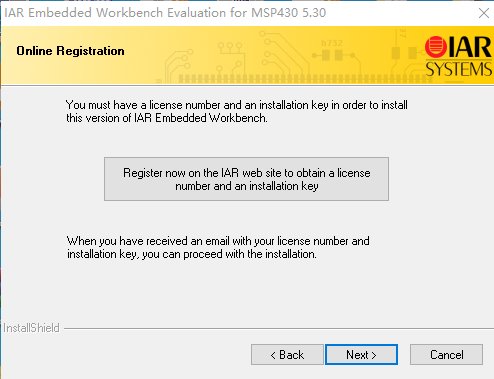
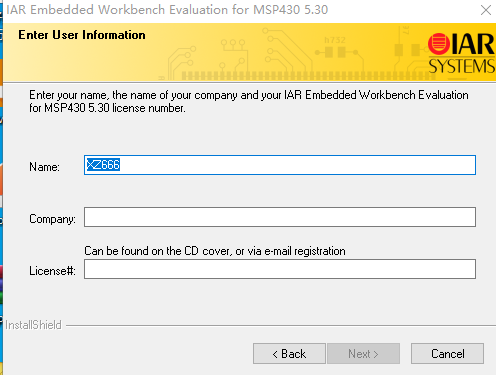
**MSP430内部flash模拟EEPROM，如何防止已经存入的数据，在第二次用JTAG下载的时候不会被清空....................................................................................................................................................42**

**IAR EW430 5.3版本IDE安装**

一定要右键管理员权限运行

解压IAR安装程序

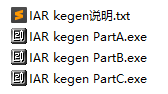
需要输入license，我们下面打开破解软件

点击next

这里不管

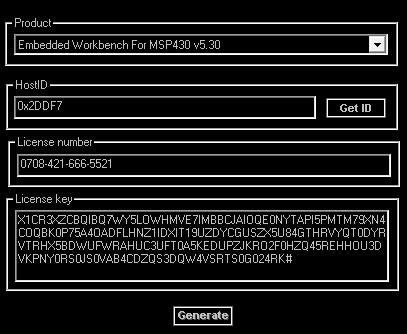
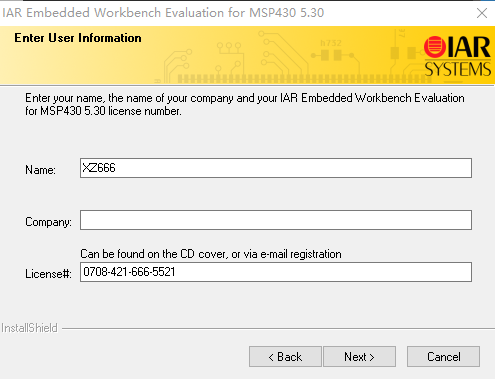
运行PartA,说明有介绍，PartA是破解MSP430和Atmel芯片的，其余partB和C是破解NEC和STM8的

以管理员方式运行

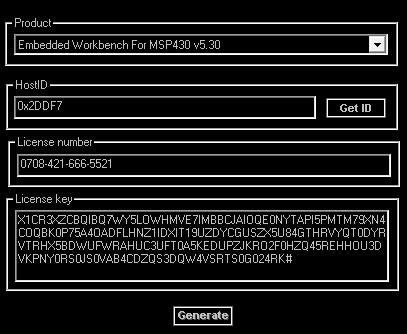
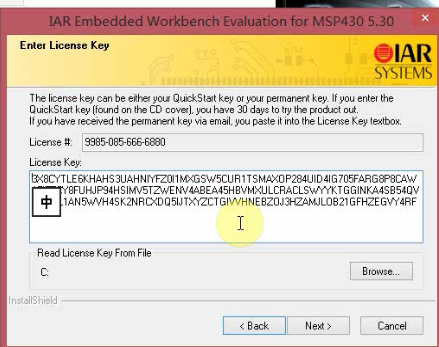
一定要先点击Get ID

解压破解软件

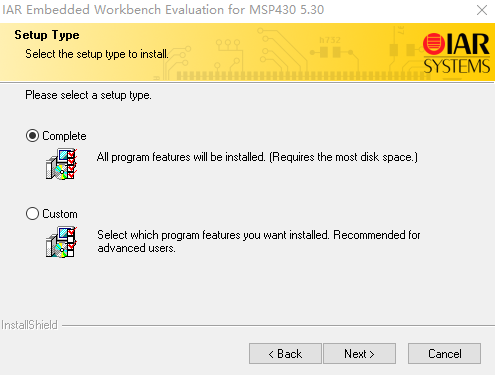
 

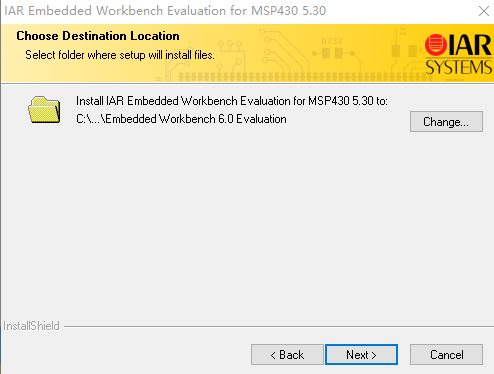
将license number复制进软件的license

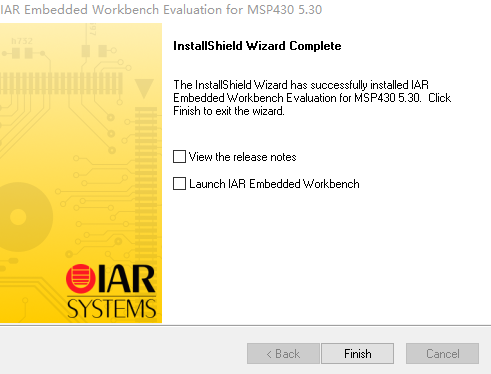
点击Next

将license key复制进软件的license key

选择Complete点击Next，然后点击install安装

我一般默认安装在C盘

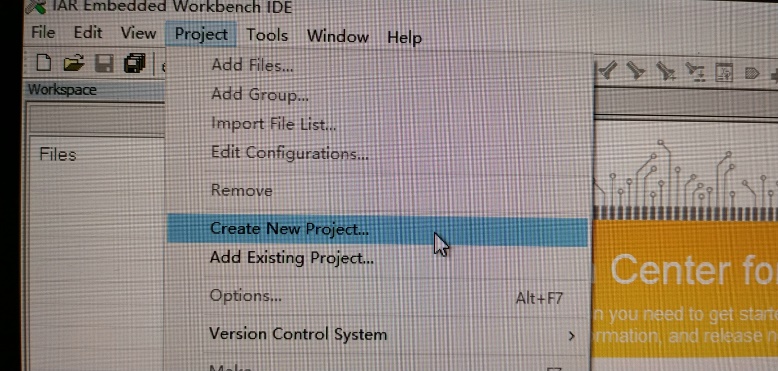
 

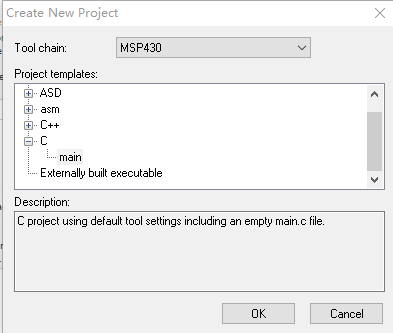
安装完的软件不会在桌面显示，一般都在软件启动菜单里显示，所以拖出快捷方式到桌面

安装完成后一定要取消这两个勾选项

 软件安装完成

IAR 创建MSP430工程

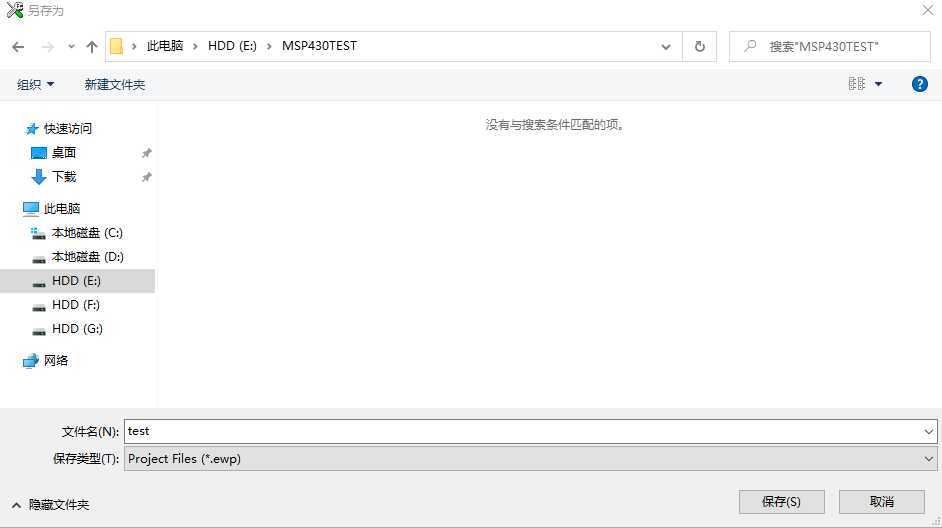
创建项目



选择MSP430

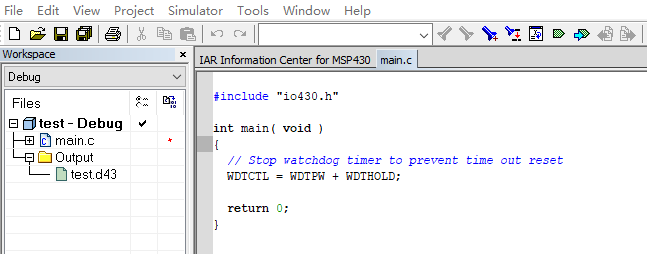
选择C文件编程

弹出保存工程路径对话框

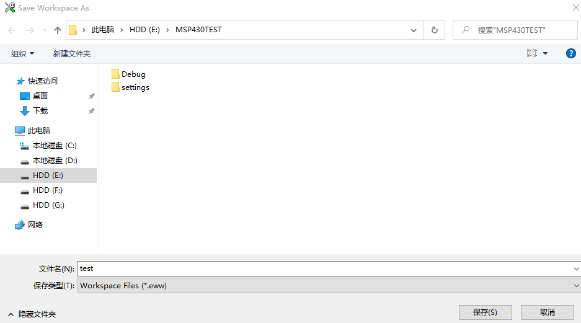
然后保存

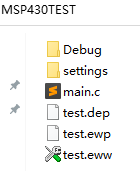
自己取工程名

一定要选择英文路径，不然会编译报错

这就是创建的基本项目结构。

因为我们是先建立的项目，而没有去先建立工作窗，所以需要执行save All保存整个项目

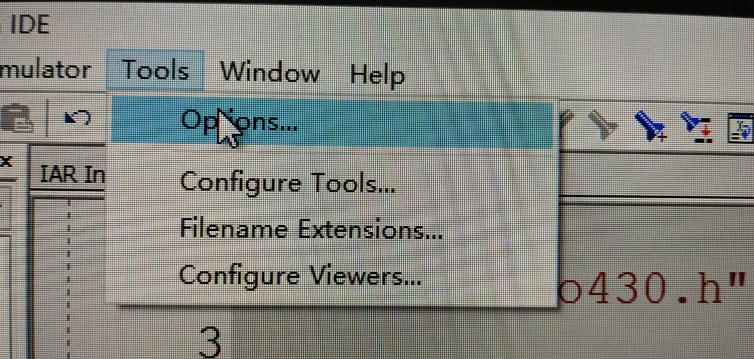
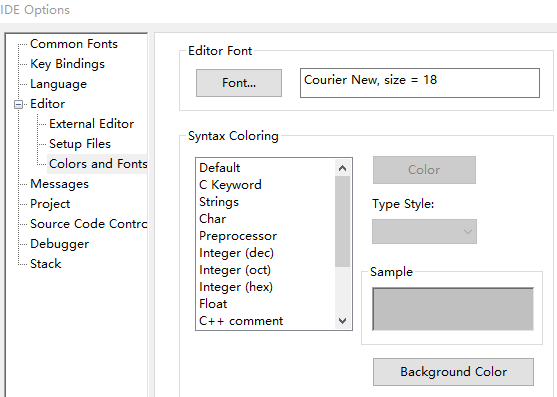
这样整个项目才算保存下来。这样才可以进行编译。

保保存项目工作窗之后，就会出现.eww文件，下次启动工程就点击.eww文件即可

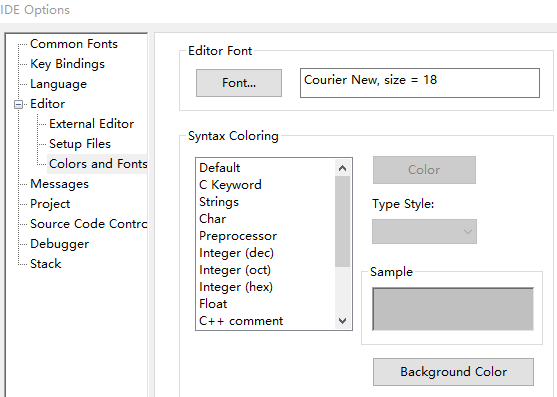
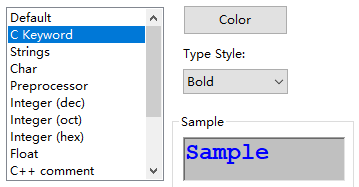
自己取项目名

IAR 字体配色方案

修改字体大小

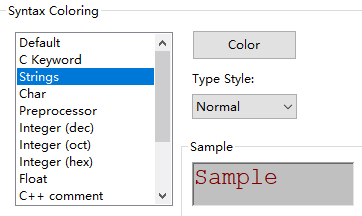
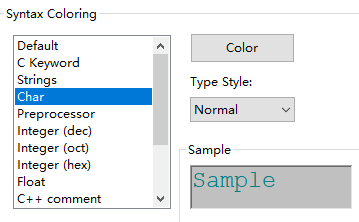
 

修改背景颜色

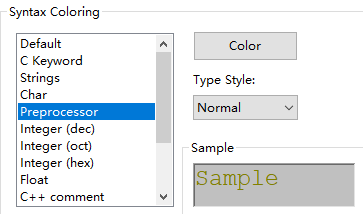
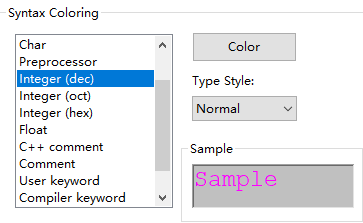
 

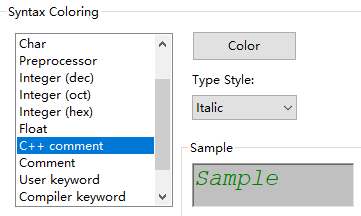
关键字用蓝色

修改编辑器C语言字符配色方案

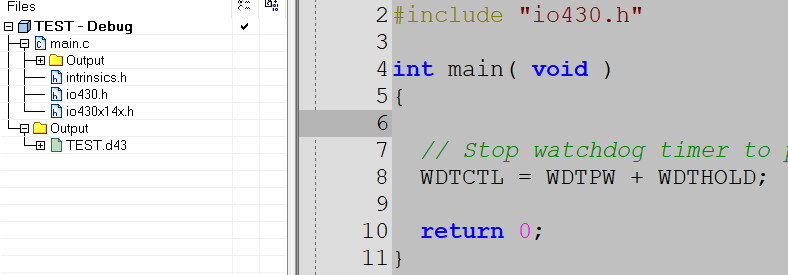
 

字符串用棕色 字符浅绿

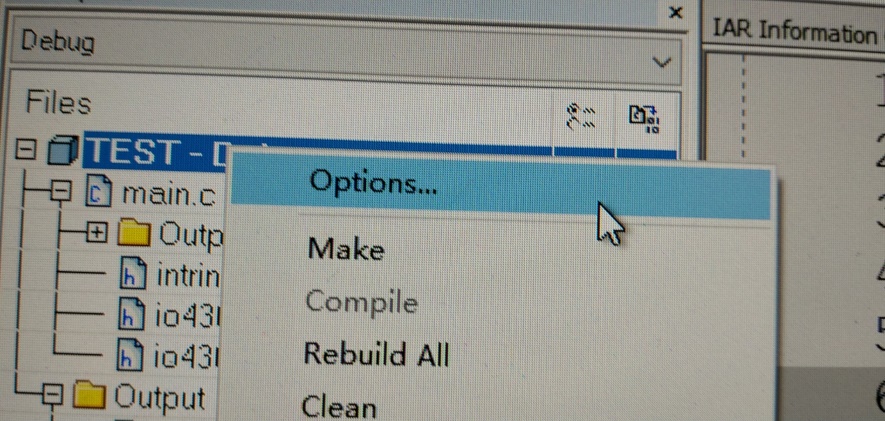
 预编译黄色十进制

注释用绿色

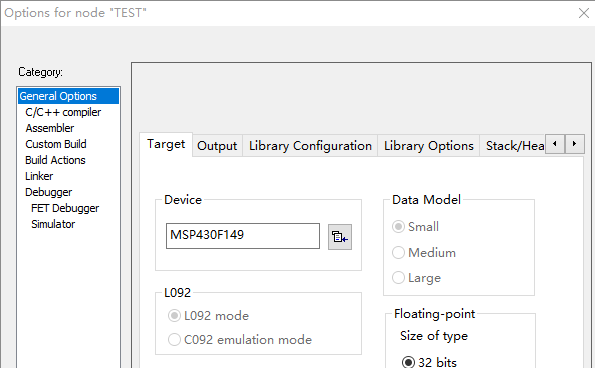
工程默认是MSP430F149,如果要用其它型号的芯片，必须在工程里面的OPTION上修改



如果要编译其它芯片型号，需要修改工程选项

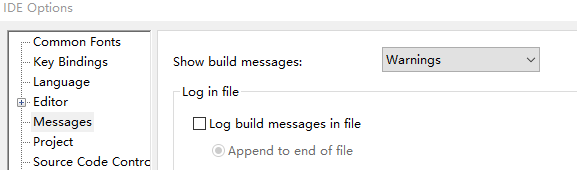


在左边工程栏右键Options

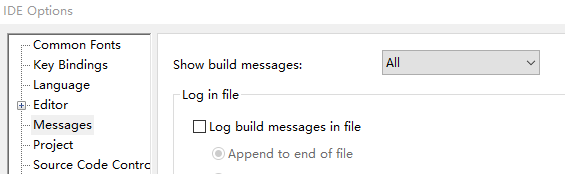


点击这儿选择其它型号

编译的时候，设置输出代码大小和所有信息



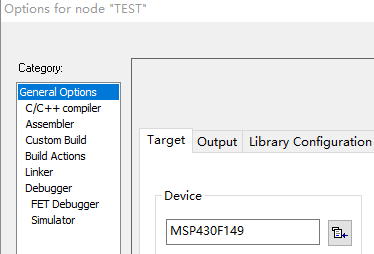
系统默认是输出警告

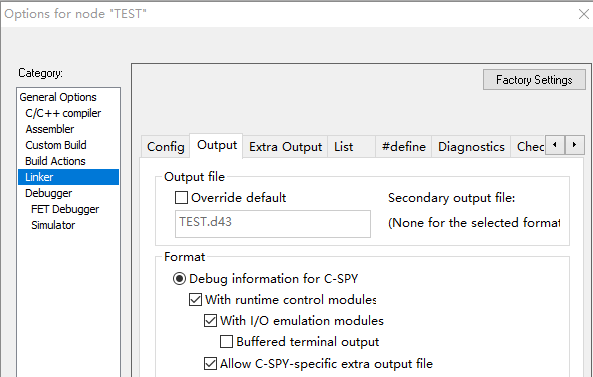
这样编译之后会输出代码大小。

输出所有

设置IAR工程输出程序供芯片下载

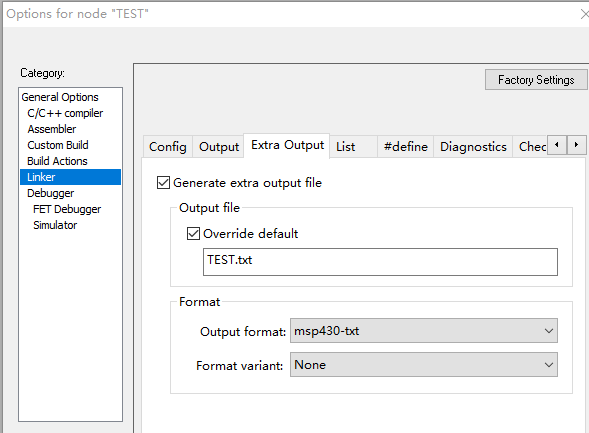
选择顶部菜单栏options





设置代码输出属性

勾选上，允许输出目标代码

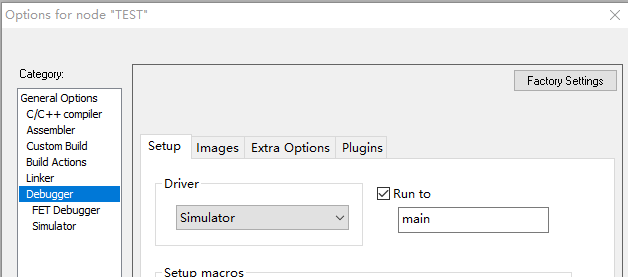


MSP430和51，STM32不一样，51和STM32的可执行程序都是hex文件，而430是txt文件下载，所以选择msp430-txt

可以设置输出txt文件的名字

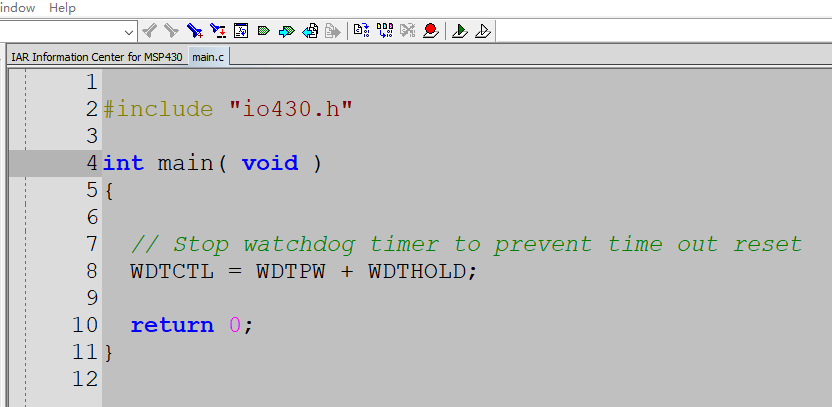
勾选上，这样每次编译出的txt程序会覆盖上一次的txt程序，这样才知道程序是否修改正确

设置输出格式

如果选择硬件仿真，还有配置其它参数，我先使用软件仿真

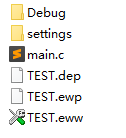
如果没有硬件仿真器，我们默认选择simulator。如果有硬件仿真器选择FET Debugger

BSL软件下载txt程序进单片机



然后再make，一定要make，make才能生成新的txt下载文件

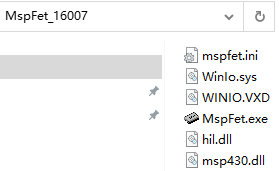
上面设置好的工程，先编译

  要下载的程序就在Debug/Exe/TEST.txt中

BSL下载程序是通过串口下载，一定要安装x64版本的CH340,或者CH341芯片驱动。

PC插上开发板有端口显示，我们就是用这个端口下载程序。

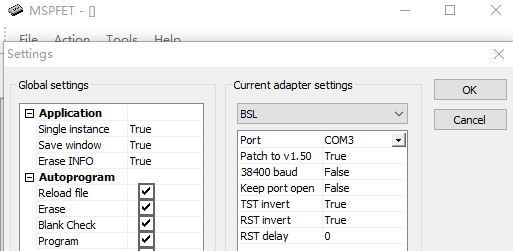
解压TI 官方的MspFet软件下载程序

主要就是运行MspFet.exe 程序，设置，下载。

最好以管理员身份运行

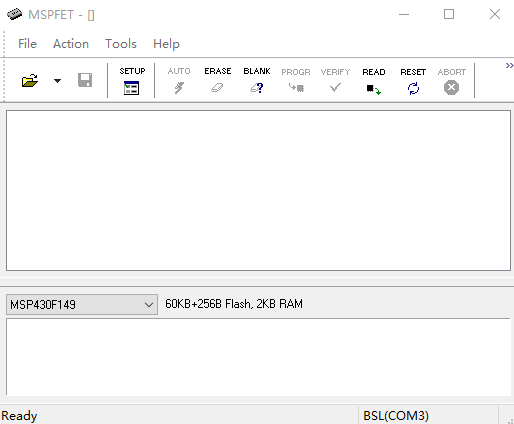


点击SETUP进行软件设置

设置完点击OK

上页我们知道CH340下载口是COM3，所以选择COM3

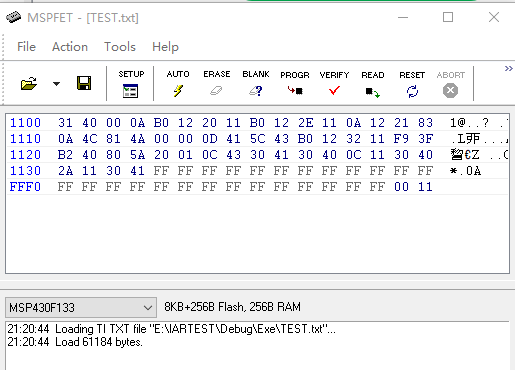
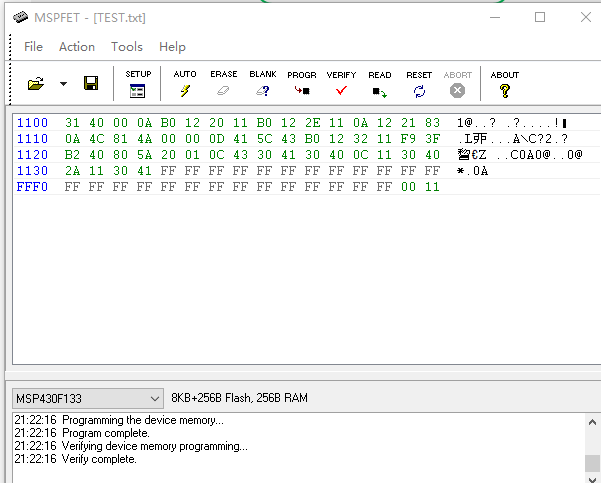
选择BSL下载



3.打开自己编译的txt文件进行下载

2.选择MSP430F149，因为要下载的芯片是F149

1.确认BSL下载端口

显示Verity complete，表示下载成功

点击AUTO下载程序，软件会自动清空芯片上一次flash，再下载

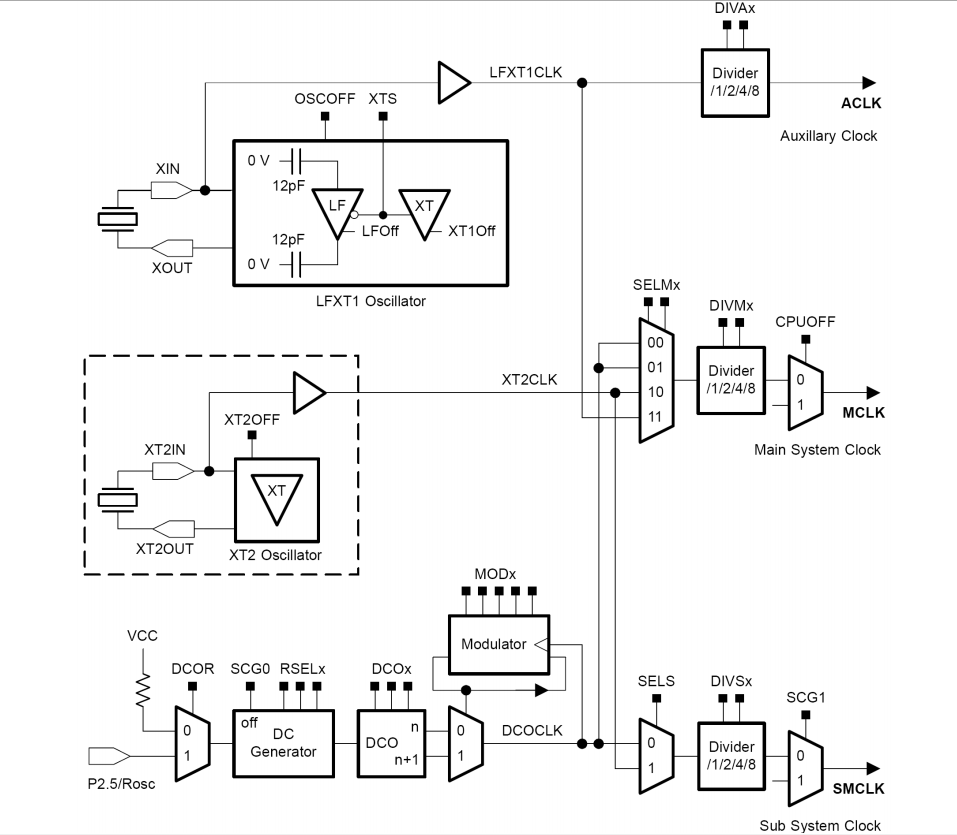
IAR 无法同时打开两个工程，第2个工程会覆盖第1个工程。如果不想第1个工程被第2个工程覆盖，就只有再次点击桌面的IAR软件，然后project->Add existing project 打开第2个工程

**时钟系统**

DCOCLK 内部时钟，如果省成本，就可以用这个DCOCLK时钟。外部晶振故障也可以切换到该时钟。

XT2CLK 标准外部时钟外接 450kHz～8MHz 的晶振工作，一般都是430的主时钟来源。

LFXT1CLK 手表时钟源，外接32.768khz，用来做时间计数。



进CPU时钟之前，先进行分频

接8M晶振

接32.768khz晶振

外部8M晶振启动，LED闪烁实验

#include "io430.h"

/\*官方库自定义延时函数\*/

#define CPU\_F ((double)8000000) //外部高频晶振为8M

//#define CPU\_F ((double)32768) //如果用32.768khz做CPU时钟，就选这句

#define delay\_us(x) \_\_delay\_cycles((long)(CPU\_F \* (double)x/1000000.0)) //除以1000000微妙

#define delay\_ms(x) \_\_delay\_cycles((long)(CPU\_F \* (double)x/1000.0)) //除以1000毫秒

BCSCTL1寄存器

0： XT2 高速晶振开

/\*时钟初始化\*/

void Clock\_Init()

{

char i;

BCSCTL1&=~XT2OFF; //打开XT2振荡器

BCSCTL2寄存器

2(10)： MCLK 时钟源为 XT2CLK

1： SMCLK 时钟源为 XT2CLK

BCSCTL2|=SELM1+SELS; //MCLK为8MHZ，SMCLK为8MHZ

do{

IFG1&=~OFIFG; //清除振荡器错误标志

for(i=0;i<100;i++)

{

}

}

while((IFG1&OFIFG)!=0); //如果标志位1，则继续循环等待

IFG1&=~OFIFG;

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

while(1)

{

P6OUT = 0xFF; //P6端口0~7个IO全部输出高电平

delay\_ms(500);

P6OUT = 0x00; //P6端口0~7个IO全部输出低电平

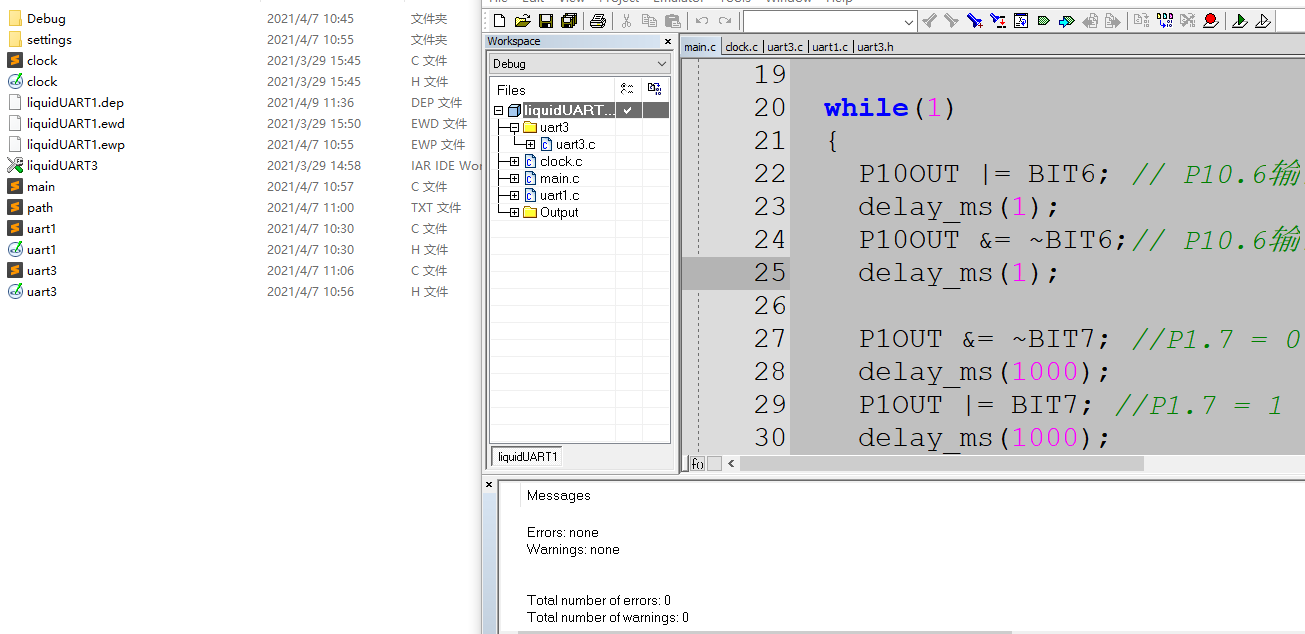
delay\_ms(500);

}

}

**IAR自定义多目录，多C/H文件路径加载方法**

所有c/h文件在同一个目录下，直接加载进工程就可以



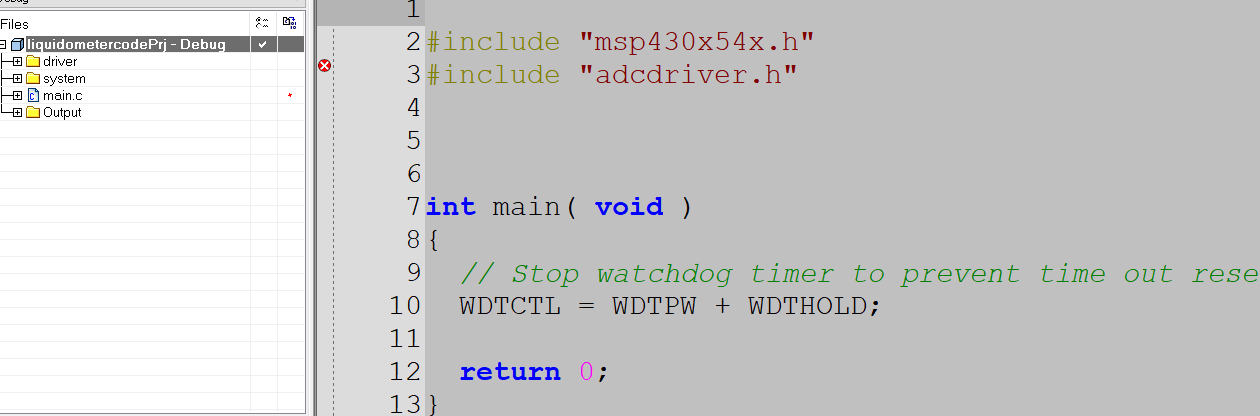
直接加入进工程没问题

所有文件都在同一个目录下

编译通过

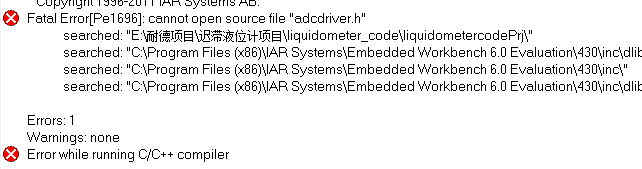


如果主函数调用的C文件在其它目录下

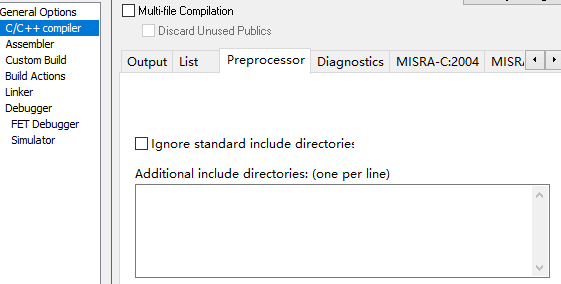


调用子目录的文件

加载子目录下的C文件

发现编译报错

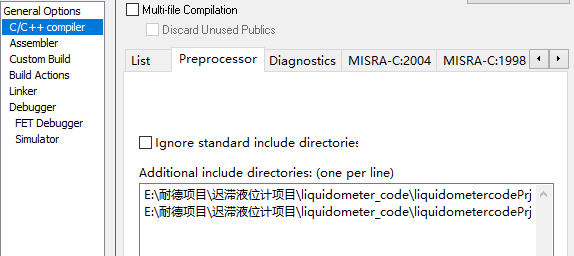
这是因为IAR 如果要加入子目录下的C/H文件，需要在IAR IDE编译器下设置文件路径



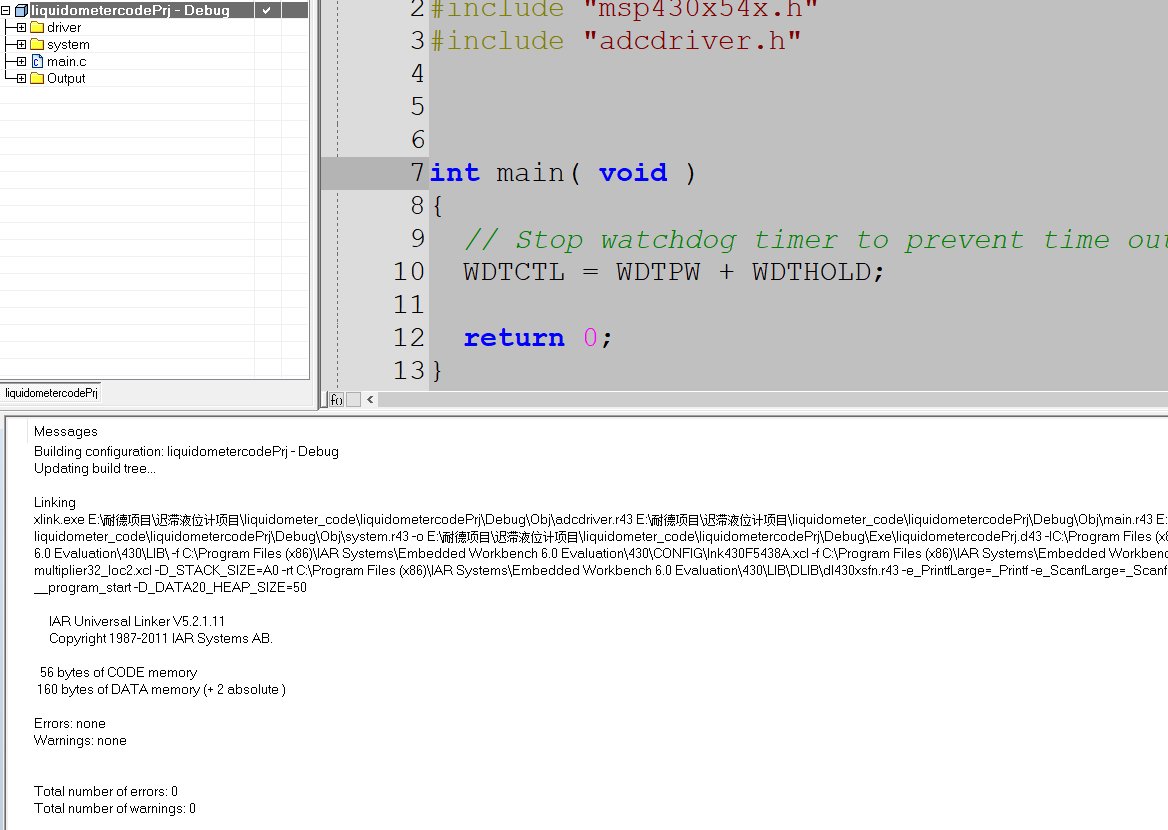
文件目录的整个路径加载这里，只需要加入目录路径就行

选择C/C++ compler

在Preprocessor下添加路径



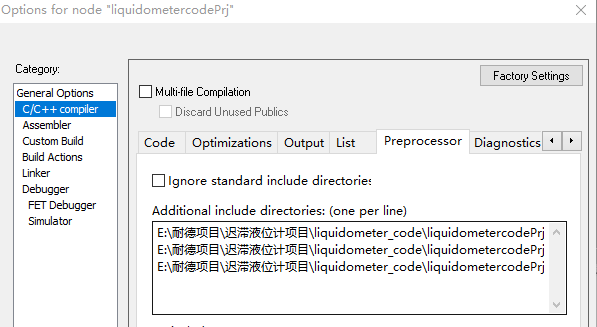
子文件目录路径加入成功



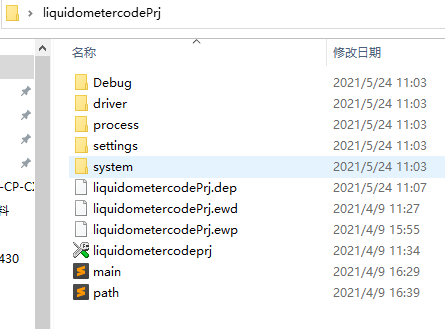
编译通过

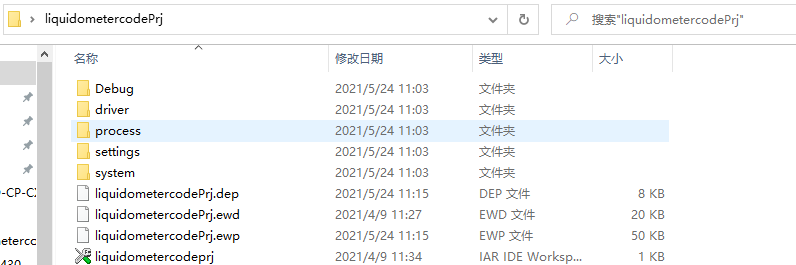
为了方便，可以使用$PROJ\_DIR$相对路径的方式来加载h文件，这样换个电脑也可以编译，不用重新修改路径

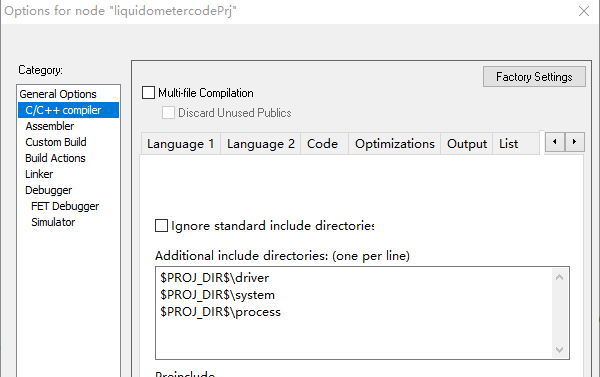
** 比如该项目拖入到桌面了**

这是liquidome以前头文件路径

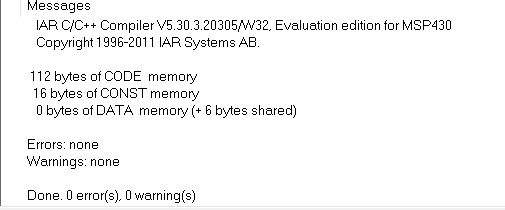
编译错误现在把绝对路径改成相对路径

头文件在第1级目录的子目录下



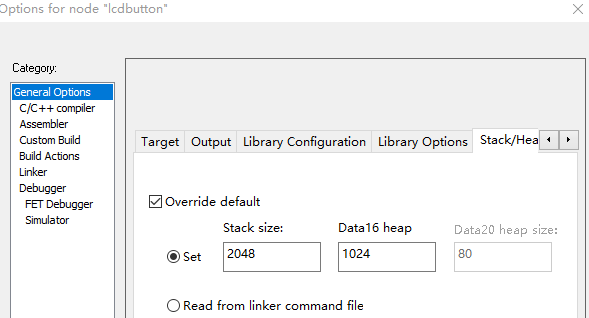


工程文件在liqudiometer…目录下，那么$PROJ\_DIR&就会指定到当前工程目录路径。后面追加的是当前工程目录下的子目录

编译成功

**IAR 设置MSP430单片机堆栈大小**

工程新建立的时候，默认堆栈是160字节

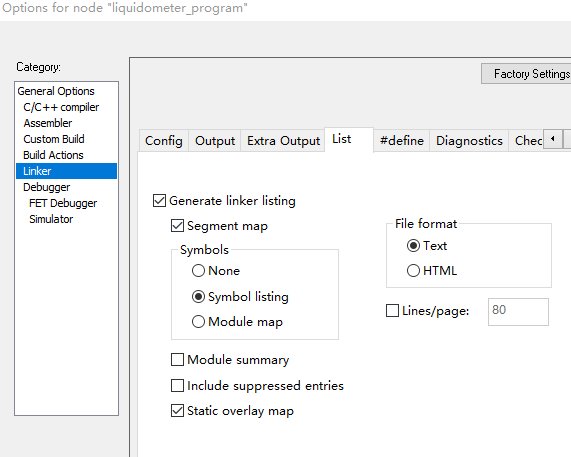


堆修改为1K

栈修改为2K

默认字节

**查看程序每个函数的栈使用大小**

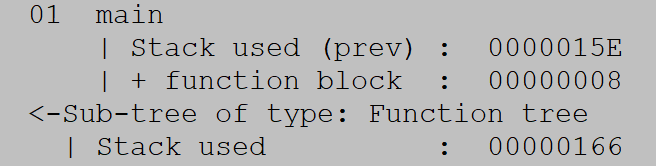


IAR设置stack输出的map文件，这样在Debug目录下就有map文件

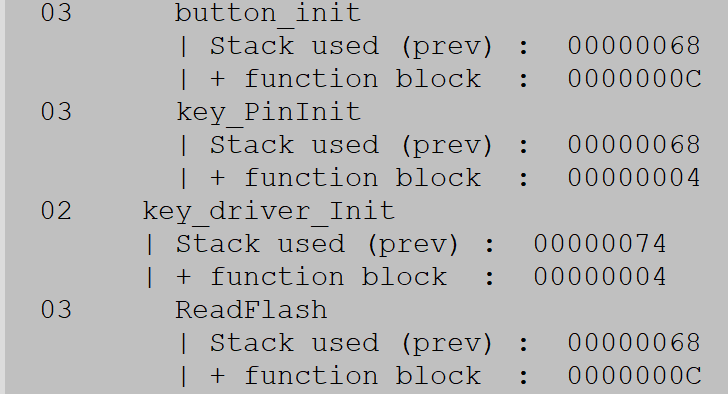




用IAR打开map文件



主函数栈使用情况



各个子函数栈使用情况

**GPIO操作**

|  |  |
| --- | --- |
| 端口 | 功能 |
| P1组、P2组 | I/O、中断能力、其它片内外设功能 |
| P2、P3、P4、P6组 | I/O、其它片内外设功能 |

P1 和 P2有 7 个寄存器

P3～P6 各有 4 个寄存器

P1 和 P2 所有 8 位全都可做外部中断处理

返回值 = PxIN\_bit.Pn 位操作变量

x: 填入第几组GPIO

n: 填入该组GPIO的第几个管脚

IO口输入输出程序实例

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

P1DIR = 0xF0; //P1.0~P1.3输入模式，外部电路已接上拉电阻

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

while(1)

{

if(P1IN\_bit.P0 == 0) //位操作(P1IN\_bit.P0) P1.0端口输入低电平，为真

{

P6OUT = 0xfe; //(1111 1110)P6.1端口 = 0 点亮LED

}

if(P1IN\_bit.P1 == 0) //位操作(P1IN\_bit.P1) P1.1端口输入低电平，为真

{

P6OUT = 0xfd; //(1111 1101)P6.2端口 = 0 点亮LED

}

if(P1IN\_bit.P2 == 0) //位操作(P1IN\_bit.P2) P1.2端口输入低电平，为真

{

P6OUT = 0xfb; //(1111 1011)P6.3端口 = 0 点亮LED

}

if(P1IN\_bit.P3 == 0) //位操作(P1IN\_bit.P3) P1.3端口输入低电平，为真

{

P6OUT = 0xf7; //(1111 0111)P6.4端口 = 0 点亮LED

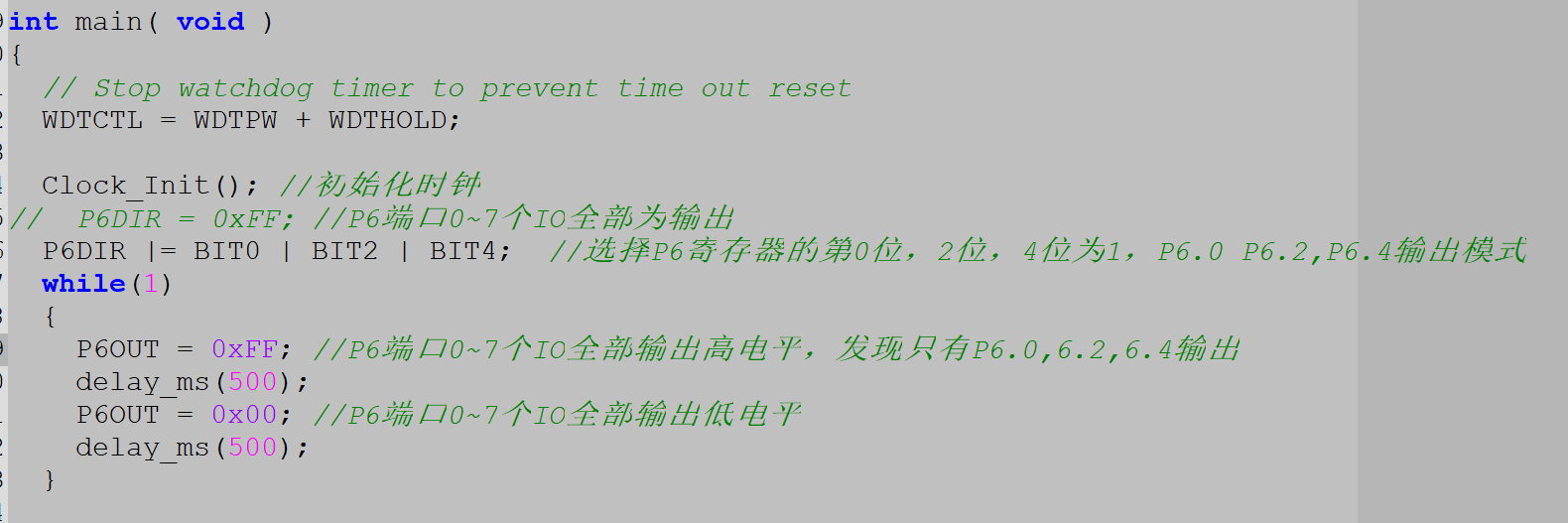
}

P6OUT = 0xFF; //P6端口0~7个IO全部输出高电平,关闭LED

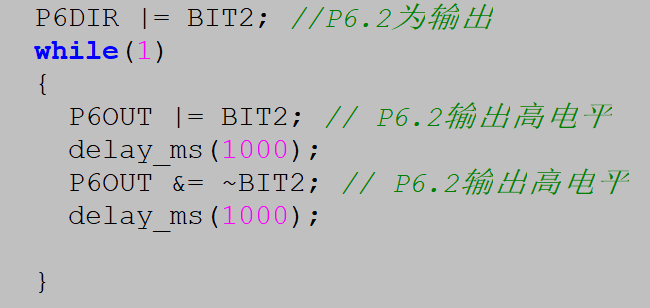
}

}

单个LED操作



单个LED输出操作



**MSP430 JTAG仿真器使用**

JTAG仿真器，TI官方自带USB供电，可以给板子供电。

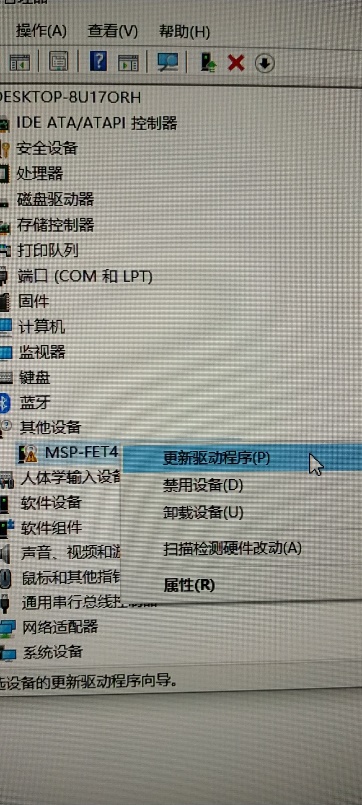
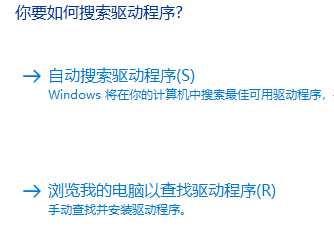
市面杂牌JTAG仿真器 有些不带给板子供电的功能

C:\Users\xiang\AppData\Local\Temp\1616117132(1).png插上仿真器后，有感叹号，表示仿真器驱动没有安装。

如果安装了IAR软件，那么在IAR软件安装目录下就有自带的驱动。我们导入即可。

C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.0 Evaluation\430\drivers\TIUSBFET\Win7-64

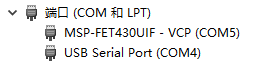
我们windows10 就使用win7-64目录来安装

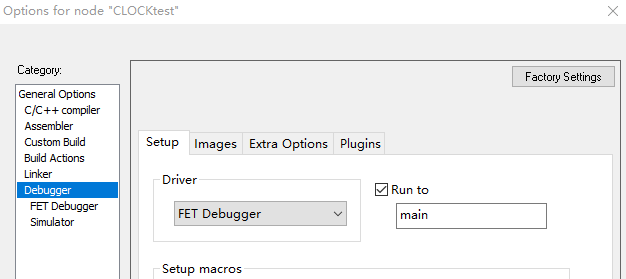
 

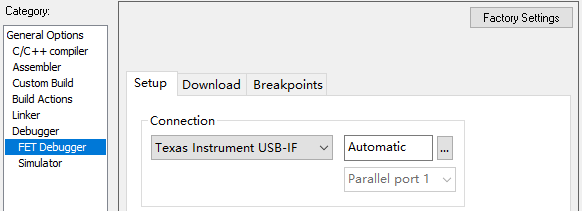
浏览系统中安装好IAR软件路径下的驱动

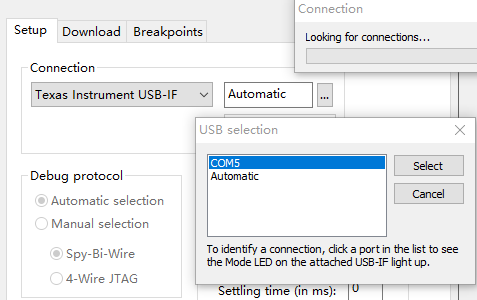


选择64位的目录

安装成功，VCP(COM5)

软件设置选择FET

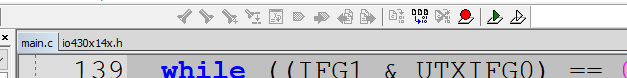
选择TI USB仿真器



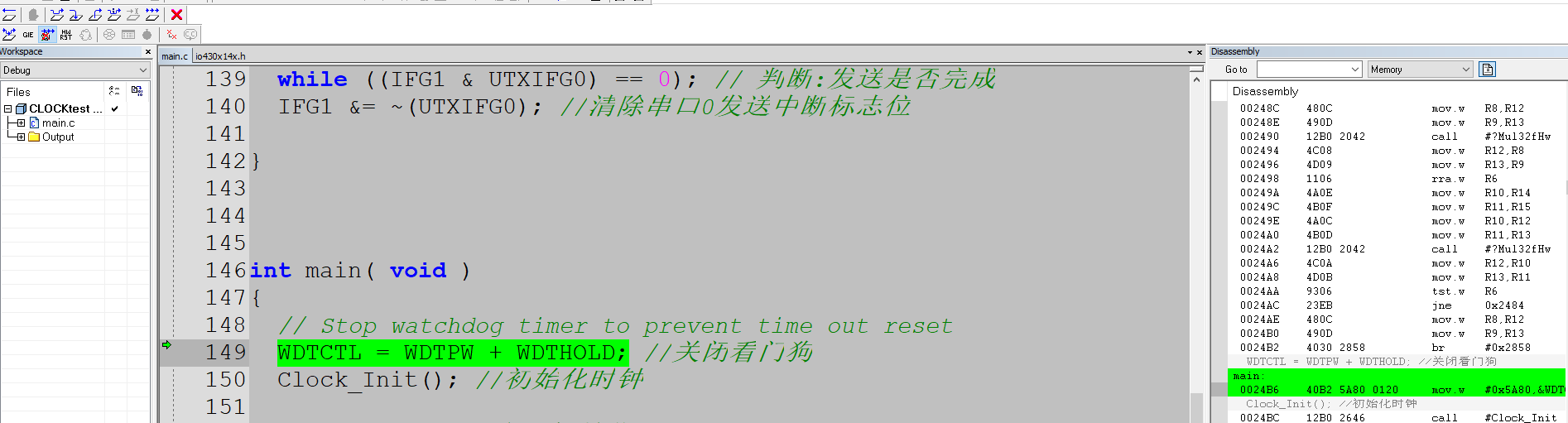
选择驱动安装成功的COM5端口，或者选择Automatic也可以

点击OK, 然后保存整个工程 Save All，硬件仿真最好使用IAR5.3版本。更高版本会有些BUG。

开始仿真



点击开始软件仿真，如果出现确定按钮，就是需要给仿真器升级，升级就是了



仿真器运行成功，就会进入调试界面，main函数第1行变成绿色

C:\Users\xiang\AppData\Local\Temp\1616120389(1).png C:\Users\xiang\AppData\Local\Temp\1616120452(1).png

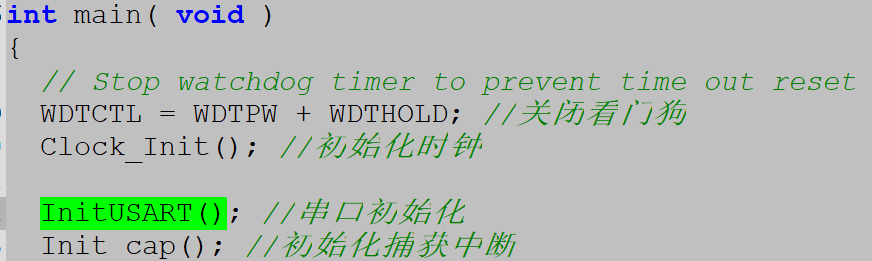
这是暂停全速运行，可以选择其它运行方式

这是退出仿真器

全速运行

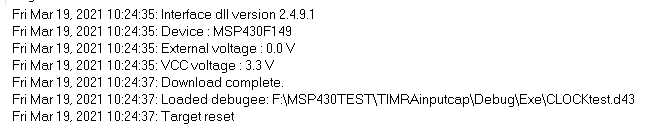
C:\Users\xiang\AppData\Local\Temp\1616120389(1).png

单步运行

单步运行了两步

如果用JTAG仿真器下载整个程序，用Download active application

选择工程Project->Download->Download active application 进行下载



下载之后显示Target reset表示下载成功。然后要点击复位键才能开始运行。

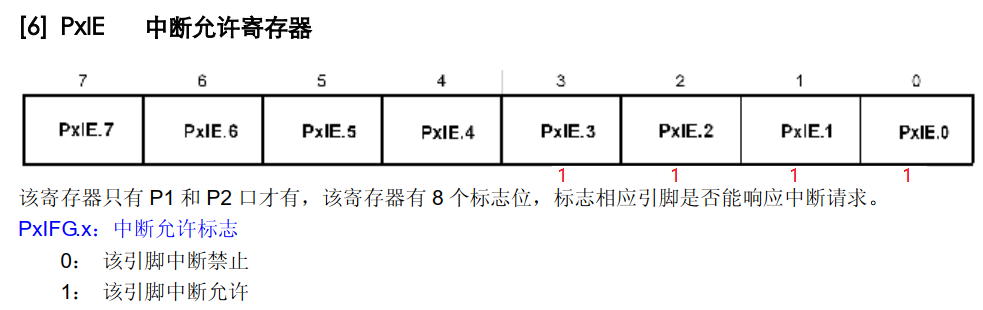
编译警告pointless integer comparison, the result is always false

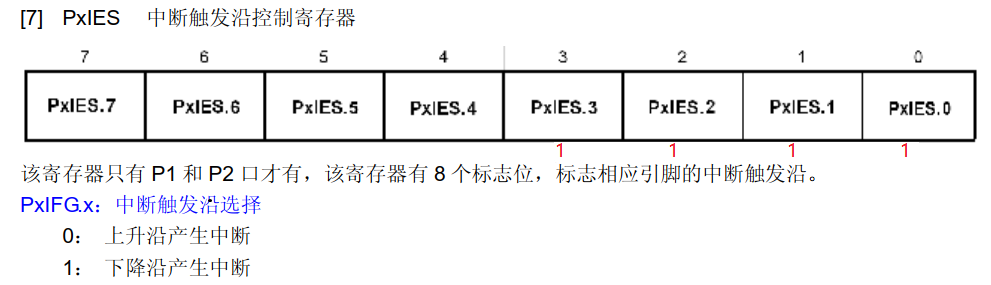
如果定义的是unsigned int 变量

然后if(变量 == 115200) 那么就会报错，因为MSP430中 unsigned int 是16位宽的最大65535

比115200少了个数量级。

**GPIO外部中断实验**





外部中断示例程序

#include "io430.h"

#include "in430.h" //外部中断\_EINT();需要用到的输入头文件

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

P1DIR = 0xF0; //P1.0~P1.3输入模式，方便做中断输入

P1IE = 0x0F; //开启P1.0~P1.3中断

P1IES = 0x0F; //P1.0~P1.3下降沿触发中断

P1IFG = 0x00; //软件清零中断标志寄存器

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

\_EINT(); //IO口寄存器设置完毕，开启总中断

while(1)

{

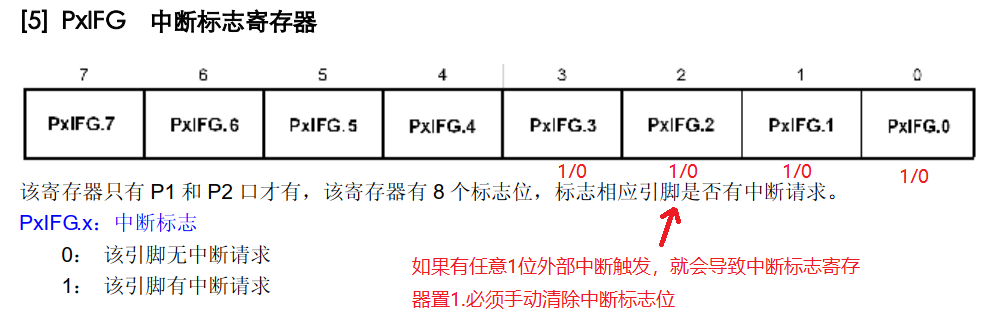
P6OUT = 0xff;

delay\_ms(1000);

}

}

/\*



\*P1口中断服务程序

\* P1组任何一个IO口中断都会进入P1中断服务程序

\*/

#pragma vector = PORT1\_VECTOR

\_\_interrupt void P1\_IRQ(void)

{

P6OUT = 0xfe; //(1111 1110)P6.1端口 = 0 点亮LED

P6OUT = 0xfd; //(1111 1101)P6.2端口 = 0 点亮LED

P6OUT = 0xfb; //(1111 1011)P6.3端口 = 0 点亮LED

P6OUT = 0xf7; //(1111 0111)P6.4端口 = 0 点亮LED

P1IFG = 0x00; //软件清零中断标志寄存器 一定要在中断中清除

delay\_ms(2000); //中断里面不要执行消耗时间的函数，这里只是为了演示LED

}

外部输入脉冲触发中断信号必须保持1.5个MCLK才能触发中断(也就是脉冲宽度必须保持1.5个MCLK)

**串口通信**

串口0程序实例

#define CPU\_F ((double)8000000) //外部高频晶振为8M

//串口波特率计算，当BRCLK=CPU\_F时用下面的公式可以计算，否则要根据设置加入分频系数

#define baud 9600 //设置波特率的大小

#define baud\_setting (unsigned int)((unsigned long)CPU\_F/((unsigned long)baud)) //波特率计算公式

#define baud\_h (unsigned char)(baud\_setting>>8) //提取高位

U0CTL寄存器

SWRST位，在初始化串口的时候一定要先将SWRST = 1，然后开始初始化串口其它参数，初始化完串口其它参数后，必须将SWRST = 0.这样才能使串口正常收发

#define baud\_l (unsigned char)(baud\_setting)

void InitUSART(void)

{

U0CTL|=SWRST; //复位SWRST

U0CTL|=CHAR; //8位数据模式

U0TCTL|=SSEL1; //SMCLK为串口时钟

U0CTL寄存器

(10) CHAR寄存器置1，8位数据位

CHAR寄存器置0，7位数据位

U0BR1=baud\_h; //SMCLK频率/9600 = 16进制值，值高8位写U0BR1

U0BR0=baud\_l; //16进制值低8位写U0BR0 注意:9600波特率8M/9600

U0MCTL=0x00; //微调寄存器为0，波特率9600bps

ME1|=UTXE0; //UART0发送使能

ME1|=URXE0; //UART0接收使能

ME1 串口0发送接受允许/禁止寄存器

U0CTL&=~SWRST; //SWRST = 0,串口初始化完成

IE1|=URXIE0; //接收中断使能位

P3SEL|= BIT4; //设置IO口为普通I/O模式

P3DIR|= BIT4; //设置IO口方向为输出

P3SEL|= BIT5;

\_EINT(); // 不要忘了开中断

}

unsigned char TBuff[8]; // 发送缓冲区

// 串口发送函数(不需要开发送中断)发送一个数组(共 8 个字节)

void USART\_Send(unsigned char \*pData)

{

unsigned char i;

for(i=0; i<8; i++)

{

TXBUF0 = pData[i]; // 装入发送寄存器

while ((IFG1 & UTXIFG0) == 0); // 判断:发送是否完成

IFG1 &= ~(UTXIFG0);

}

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

unsigned char buffer[8] = {0x01,0x02,0x03,0x04,0x05,0x06,0x07};

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

P6OUT = 0x00; //点亮全部LED

while(1)

{

USART\_Send(buffer);

delay\_ms(1000);

}

}

串口printf实现

只需要重新编写putchar函数即可

#include <stdio.h>

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化 9600波特率

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

P6OUT = 0x00; //点亮全部LED

int data = 20;

while(1)

{

printf("xxxzzzzz = %d\n",data); //IAR 430编辑器换行只需要’\n即可’不需要\r

delay\_ms(1000);

}

}

/\*

\* 主要实现printf底层的putchar函数重写

\*/

int putchar(int ch)

{

if(ch == '\n')

{

while ((IFG1 & UTXIFG0) == 0); // 判断:发送是否完成

TXBUF0 = 0x0d; // 装入发送寄存器

}

while ((IFG1 & UTXIFG0) == 0); // 判断:发送是否完成

TXBUF0 = ch; // 装入发送寄存器

return (ch);

}

串口0接受中断实现

/\*

\* 串口单字节发送

\*/

void USART\_Send(unsigned char pData)

{

TXBUF0 = pData; // 装入发送寄存器

while ((IFG1 & UTXIFG0) == 0); // 判断:发送是否完成

IFG1 &= ~(UTXIFG0); //清除串口0发送中断标志位

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

while(1)

{

delay\_ms(1000);

}

}

// 串口接受函数(需要开接受中断)

#pragma vector=UART0RX\_VECTOR

\_\_interrupt void USART0\_RXIRQ (void)

{

unsigned char Temp;

// 暂存接受数据

Temp = RXBUF0;

IFG1 &= ~(URXIFG0); //清除串口0接收中断标志位

USART\_Send(Temp);//接受数据转发回PC

}

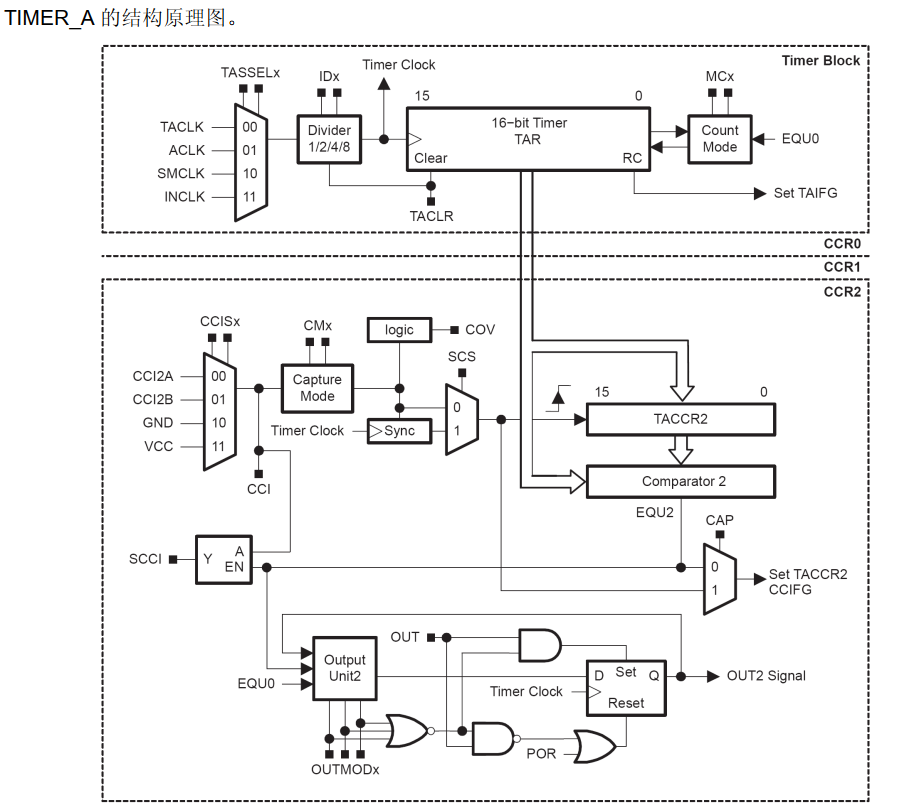
**16位定时器A使用**

定时器A有3个捕获比较寄存器，可配置位PWM输出

支持16位异步定时计数

PWM输出配合滤波器输出DAC

对外部输入中断进行计数

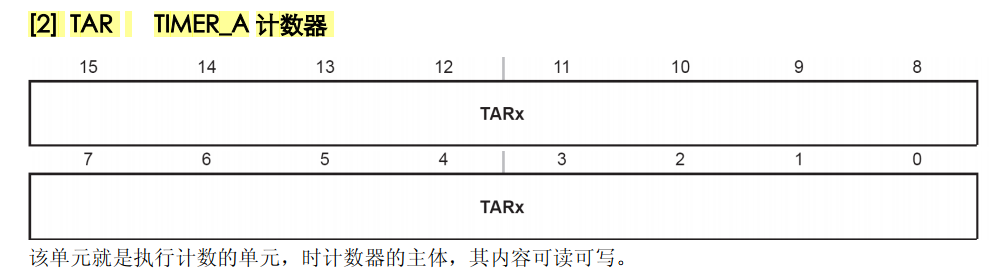


如果我们要设置定时时间，就把这个定时值写入TACCR0/2当中，当计数器计数值等于TACCR0/2的值时，产生中断信号

计数模式有4种可以设置

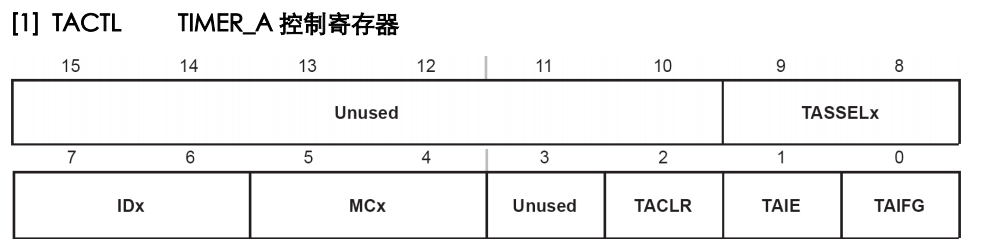
ACLK 32.678K输入

SMCLK外部8M晶振输入



TAR计数器，如果不和TACCR0对比，那么TAR 16位计数器计数满0xffff就会产生溢出中断。

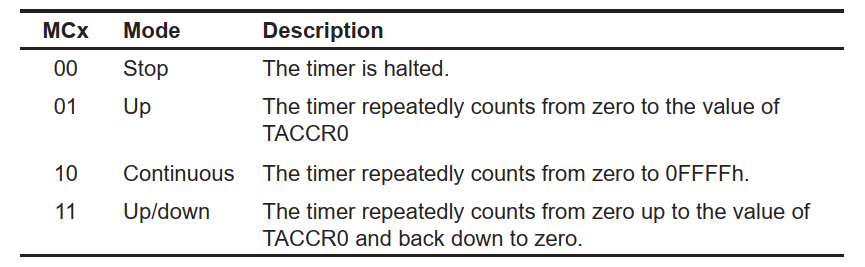
TAR计数器和TACCR0对比，那么TAR计数计数值增加到与TACCR0值一样时，会产生捕获比较中断。



TACTL寄存器，要注意TACLR位清0功能。该位不仅会清除计数器的值，还会清除定时器时钟的设置。

还会清除计数器递增，递减方向设置。

定时器启动方式



MCx位 >0 时，并且定时器外部输入的时钟源有效。那么定时器就开始启动。所以这一点和STM32和51还不一样。MSP430没有专门的定时器启动函数。

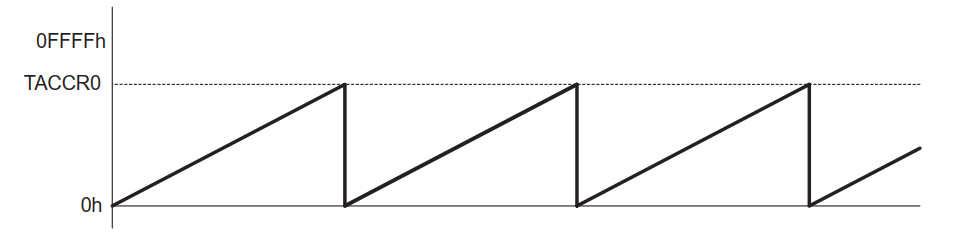
如果MCx是计数器递增或递减模式。通过给TACCR0写0就可以终止计数。如果给TACCR0写非0的值，计数器就会从0开始重新启动。一直计数到TACCR0 产生中断。

Mcx 为00 (停止模式)，用于定时器暂停，但是不会给计数器清0，当定时器从暂停之后开始启动。那么计数器会接着计数。类似播放音乐这种暂停方式。而且最先设置的计数器递增或者递减模式不会被改变。如果想定时器暂停后改变计数器的模式。就必须给TACCR0清0

或者给TACLR位清0，来清除设置。然后重新设置定时器。

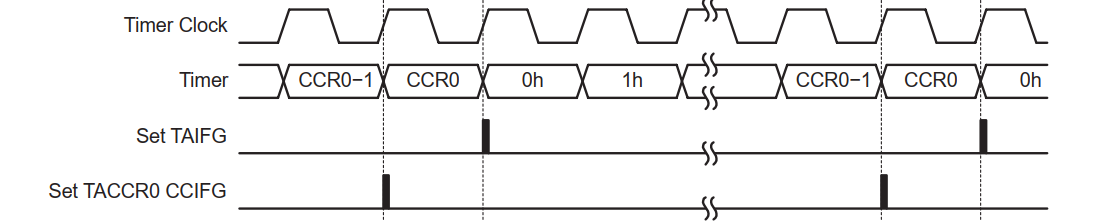
Mcx 为01(UP模式): 计数器递增重复计数，从0开始计数到TACCR0值。TACCR0就是设置的定时时间

产生捕获比较中断



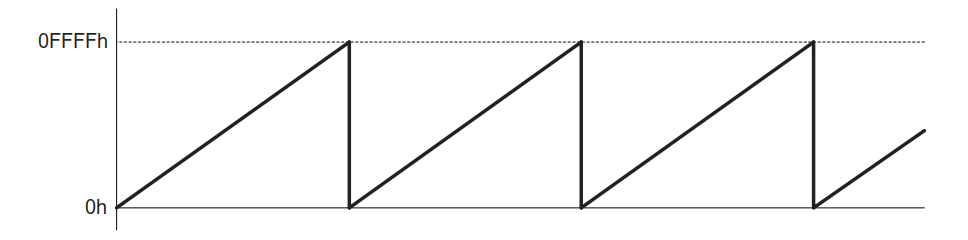
产生溢出中断

这就是up模式,计数器计数到TACCR0时，产生捕获比较中断，然后计数器又重新从0开始计数。(注意计数器变为0时会产生溢出中断)。所以up模式是两个中断。

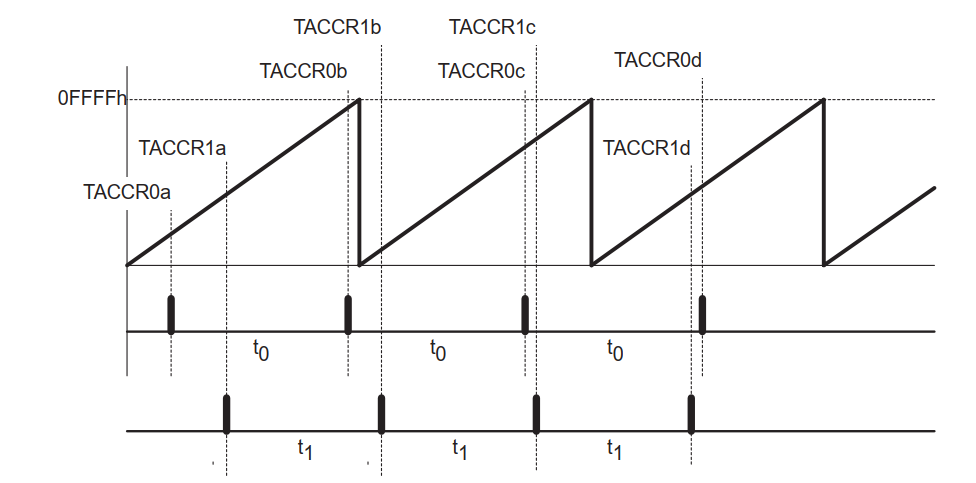


这就是up模式的两个中断。

Mcx 为10(连续计数模式): 就是计数器最大值为0xffff，不能修改，计数器不停的计数，计数到0xffff产生中断，然后计数器清0，从新开始计数。



我们也可以利用TACCR0,TACCR1来对连续模式的定时器进行采样。这样可以产生多个不同计数值得中断

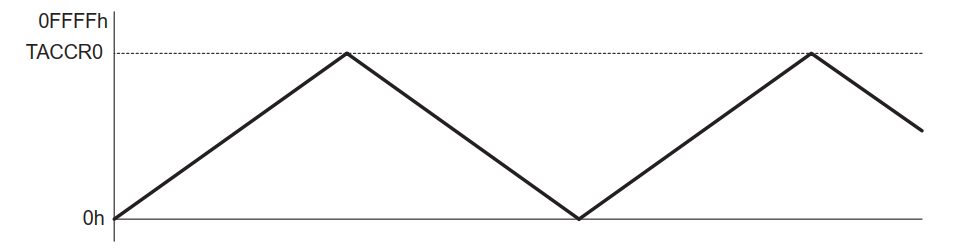


TACCR1要求的计数器中断值

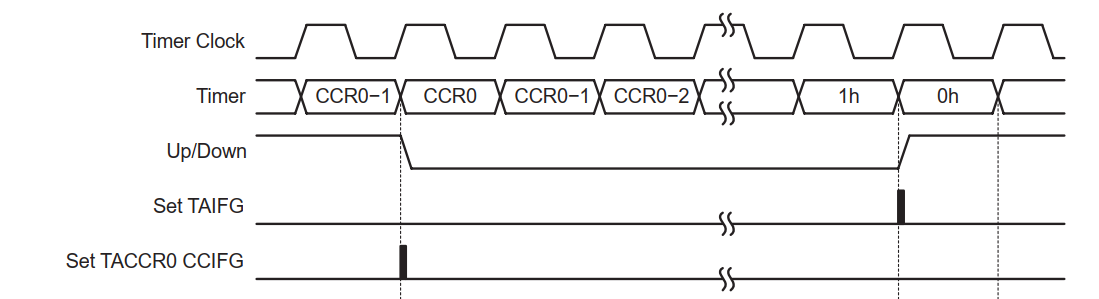
TACCR0要求的计数器中断值

这种方式主要是使用外部引进捕获功能。来计算外部信号的时间宽度。

Mcx 为11(连续递增递减模式): 计数器先从0递增到TACCR0，然后TACCR0再递减到0，重复执行

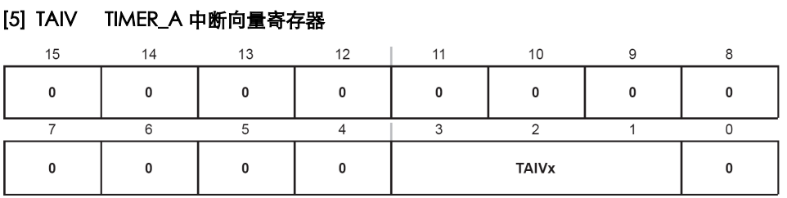


适合做三角波发生器。

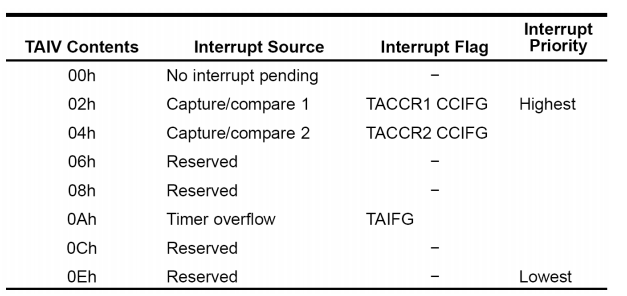


递增递减模式，一个三角波两个中断。

在定时器中断函数里面要判断是哪一个中断向量触发了



定时器中断函数里面就是查询TAIV来确定是什么哪一类中断



02,04是比较捕获中断，0A是定时器溢出中断

定时器连续模式闪烁LED实例

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 定时器A初始化

\* 定时器连续模式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TIMRA(void)

{

TACTL |= TASSEL1 + TACLR + ID0 + ID1 + MC1 + TAIE;

//TASSEL1 用SMCLK做时钟源，

//TACLR 清除定时器默认配置

//ID0 + ID1 (11) 8分频(晶振8M/8=1M)计数器加1个数是1us

//MC1连续计数模式，计数到0XFFFF

//TAIE (10) 允许定时器溢出中断,清除TA溢出标志位

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

TIMRA(); //定时器初始化

\_EINT(); //开启总中断

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

P6OUT = 0x00; //点亮全部LED

while(1)

{

}

}

/\*定时器A中断触发程序\*/

#pragma vector = TIMERA1\_VECTOR

\_\_interrupt void Timer\_A(void)

{

//定时器的中断，必须读取TAIV变量来清除中断标志位，

//不然第一次进入中断后就无法退出中断，一定要注意，困扰了我一下午

switch(TAIV)

{

case 10: P6OUT = ~P6OUT;break; //10(0A) 定时器溢出中断,65ms中断一次，连续模式是增加到65535(0xffff)才会中断，LED快闪。

}

}

定时器指定，定时时间实例

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 定时器A初始化

\* 定时器增量模式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TIMRA(void)

{

TACTL |= TASSEL1 + TACLR + ID0 + ID1 + MC0 + TAIE;

//TASSEL1 用SMCLK做时钟源，

//TACLR 清除 TAR，时钟分频，计数模式的设置。清除设置后自动清零

//MC0(01) 计数器增量模式

//ID0 + ID1 (11) 8分频(晶振8M/8=1M)计数器加1个数是1us

//TAIE 允许定时器溢出中断

TACCR0 = 9999; //定时时间初值，计数器1us加一次，10ms产生一次中断

//等待计数器累加到TACCR0的值，就产生中断

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

TIMRA(); //定时器初始化

\_EINT(); //开启总中断

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

P6OUT = 0x00; //点亮全部LED

while(1)

{

if(Count > 20) //这就是定时时间，定时器10ms中断一次，200ms间隔灯翻转

{

Count = 0;

P6OUT = ~P6OUT;

}

// delay\_ms(500);

}

}

/\*定时器A中断触发程序\*/

#pragma vector = TIMERA1\_VECTOR

\_\_interrupt void Timer\_A(void)

{

//定时器的中断，必须读取TAIV变量来清除中断标志位，

//不然第一次进入中断后就无法退出中断，一定要注意，困扰了我一下午

switch(TAIV)

{

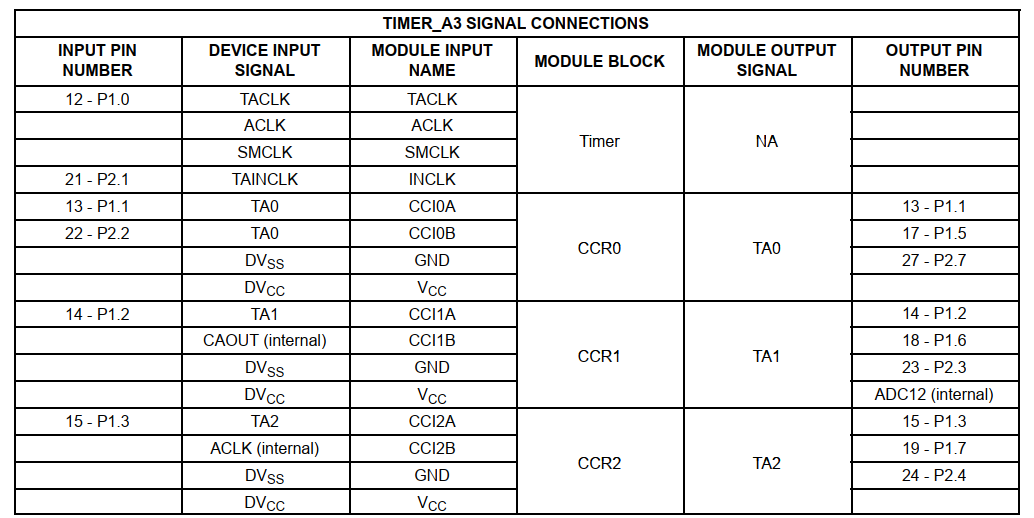
case 10:Count++ ;break; //10(0A)定时器溢出中断

}

}

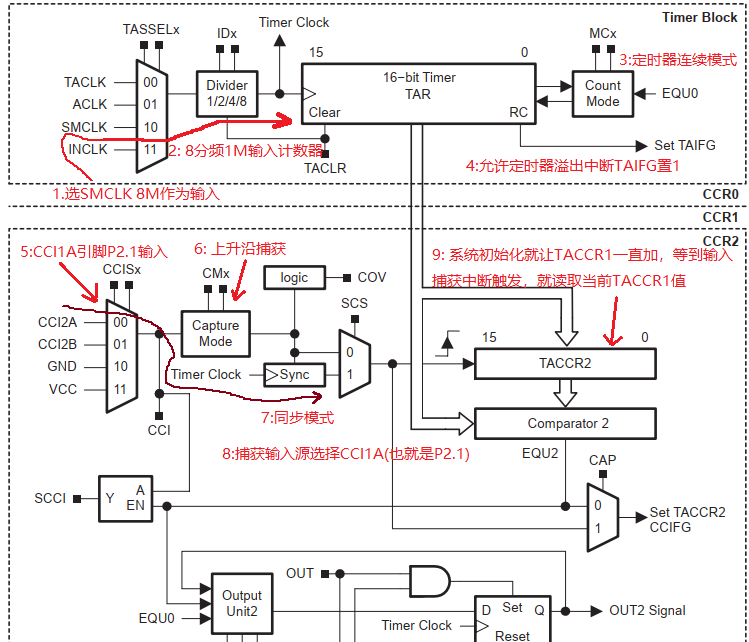
引脚输入信号电平捕获(定时器输入捕获)

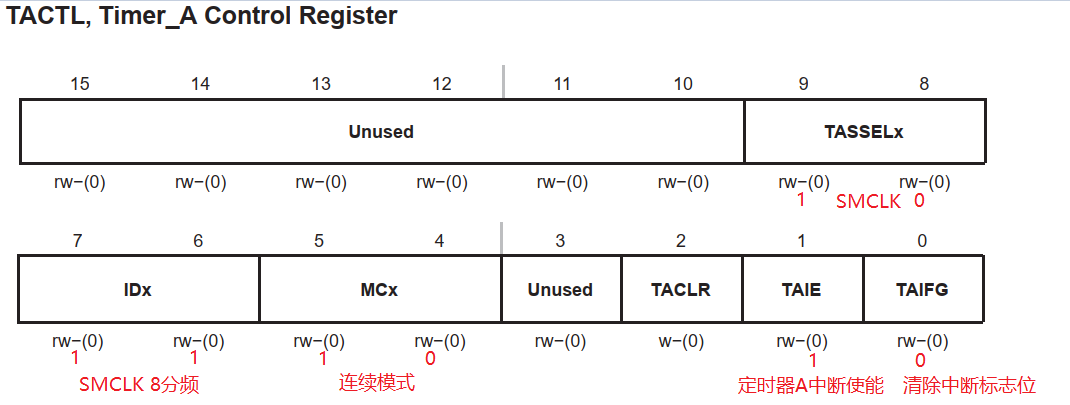
输入捕获引脚有哪些?



我们选用P1.2端口输入捕获，那么就是TIMRA定时器，获取TACCR1寄存器数据计算方波周期

方波周期采集





/\*

\* 定时器A初始化

\*/

void Init\_cap(void)

{

P1DIR = 0x00; //P1.2设置IO口为输入模式

P1SEL |=0x04; //选择P1.2作为输入捕获功能

TACTL =TASSEL\_2+MC\_2+ID\_3+TAIE; //选择8M－SMCLK时钟

//TASSEL\_2 选择SMLCK时钟

//MC\_2 定时器连续模式

//ID\_3输入时钟8分频，SMLCK/8=1M(1us) TIMRA 定时器的计数器1us累加1次

// TAIE 允许定时器溢出中断

TACCR1寄存器什么时候启动?

当外部引脚有输入边沿信号触发捕获的时候,将定时器TAR这时候累加的值复制给TACCR1寄存器。所以TACCR1寄存器并没有去根据时钟累加值，而是用的TAR的值

TACCTL1 = CM\_1+SCS+CCIS\_0+CAP+CCIE; //上升沿触发，同步模式，使能中断

//CM\_1 输入捕获上升沿采样，捕获到输入方波上升沿 进入0x02捕获比较1中断

//SCS 同步捕获

//CCIS\_0 输入捕获引脚选择CCI1A

//CAP 输入捕获模式

//CCIE 允许TACCR1捕获中断

TACCR1 = 0; //清0 TACCR1寄存器

}

unsigned int old\_cap = 0; //上一次的捕获计数值

unsigned int period = 0; //得到信号周期

unsigned int TA\_ov\_num = 0; //定时器计数溢出次数

/\*

\* 定时器A连续模式，计数器溢出中断

\* CCI1A输入捕获中断

\*/

#pragma vector=TIMERA1\_VECTOR

\_\_interrupt void Timer\_A(void)

{

switch(TAIV)

{

case 2: //0x02 捕获比较1中断

period = TA\_ov\_num\*65536+TACCR1-old\_cap; //计算周期宽度，单位us，这是1M SMCLK时钟下，如果是2M时钟那就用0.5us计算

old\_cap = TACCR1;

TA\_ov\_num = 0;

break;

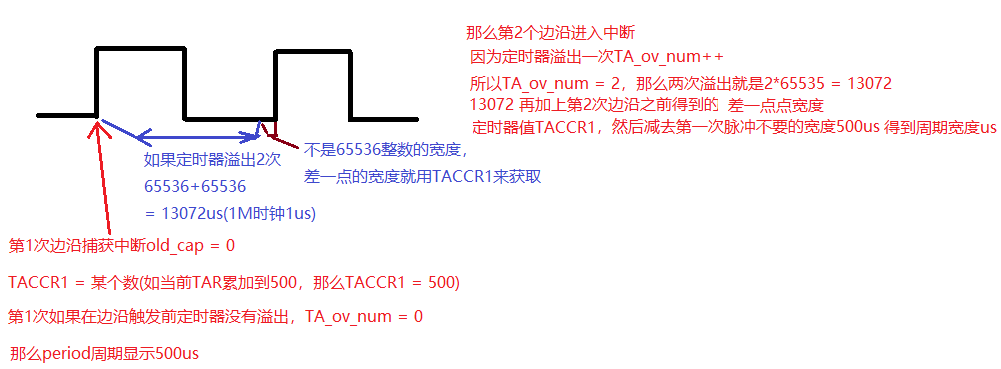
case 4:break;

case 10:TA\_ov\_num++; //0x0A(10),定时器溢出中断

break;

}

}



int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

Init\_cap(); //初始化捕获中断

\_EINT(); //开启总中断

while(1)

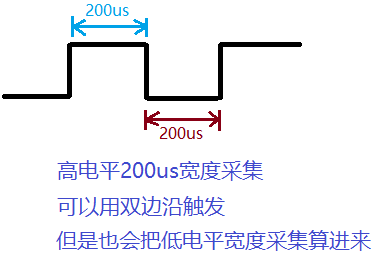
{

printf("period = %d\n", period);

}

}

方波高电平时间采集



unsigned int old\_cap = 0; //上一次的捕获计数值

unsigned int HightLevelWidht = 0;//得到信号高电平宽度

unsigned int TA\_ov\_num = 0; //定时器计数溢出次数

/\*

\* 定时器A初始化

\*/

void Init\_cap(void)

{

P1DIR = 0x00; //P1.2设置IO口为输入模式

P1SEL |=0x04; //选择P1.2作为输入捕获功能

TACTL =TASSEL\_2+MC\_2+ID\_3+TAIE; //选择8M－SMCLK时钟

//TASSEL\_2 选择SMLCK时钟

//MC\_2 定时器连续模式

//ID\_3输入时钟8分频，SMLCK/8=1M(1us) TIMRA 定时器的计数器1us累加1次

// TAIE 允许定时器溢出中断

TACCTL1 = CM\_3+SCS+CCIS\_0+CAP+CCIE;//上升沿触发，同步模式，使能中断

//CM\_3 输入捕获双边沿采样，捕获到输入方波上升沿或者下降沿 进入0x02捕获比较1中断，主要修改这句

//SCS 同步捕获

//CCIS\_0 输入捕获引脚选择CCI1A

//CAP 输入捕获模式

//CCIE 允许TACCR1捕获中断

TACCR1 = 0; //清0 TACCR1寄存器

}

/\*

\* 定时器B初始化

\*/

void Init\_timer\_b(void) //为了不影响主循环模拟方波输出，打印交给定时器B

{

TBCCTL0 = CCIE; // CCR0 interrupt enabled

TBCCR0 = 32768; //定时1秒

TBCTL = TBSSEL\_1 + MC\_1; // ACLK, upmode

}

/\*

\* 定时器A连续模式，计数器溢出中断

\* CCI1A输入捕获中断

\*/

#pragma vector=TIMERA1\_VECTOR

\_\_interrupt void Timer\_A(void)

{

switch(TAIV)

{

case 2: //0x02 捕获比较1中断

HightLevelWidht = TA\_ov\_num\*65536+TACCR1-old\_cap;

old\_cap = TACCR1;

TA\_ov\_num = 0;

break;

case 4:break;

case 10:TA\_ov\_num++; //0x0A(10),定时器溢出中断

break;

}

}

#pragma vector=TIMERB0\_VECTOR

\_\_interrupt void Timer\_B (void)

{

printf("Widht = %d\n",HightLevelWidht); //用来做1秒的方波电平宽度打印

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

Init\_cap(); //初始化捕获中断

Init\_timer\_b(); //初始化定时器B CCR0中断

\_EINT(); //开启总中断

P6DIR = 0xFF; //P6端口0~7个IO全部为输出

P6OUT = 0x00; //点亮全部LED

while(1)

{

/\*IO口模拟输出200us的高电平，200us低电平方波\*/

P6DIR = 0xFF; //P6组全输出高电平

delay\_us(200);

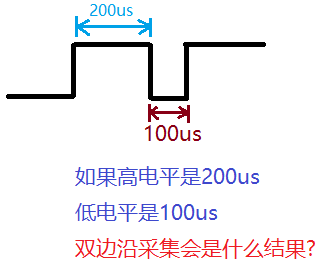
P6DIR = 0x00; //P6组全输出低电平

delay\_us(200);

}

}

 207us 高电平宽度和低电平宽度。主要是驱动了LED所以有几个us误差。



while(1)

{

/\*IO口模拟输出200us的高电平，100us低电平方波\*/

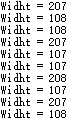
P6DIR = 0xFF; //P6组全输出高电平200us

delay\_us(200);

P6DIR = 0x00; //P6组全输出低电平100us

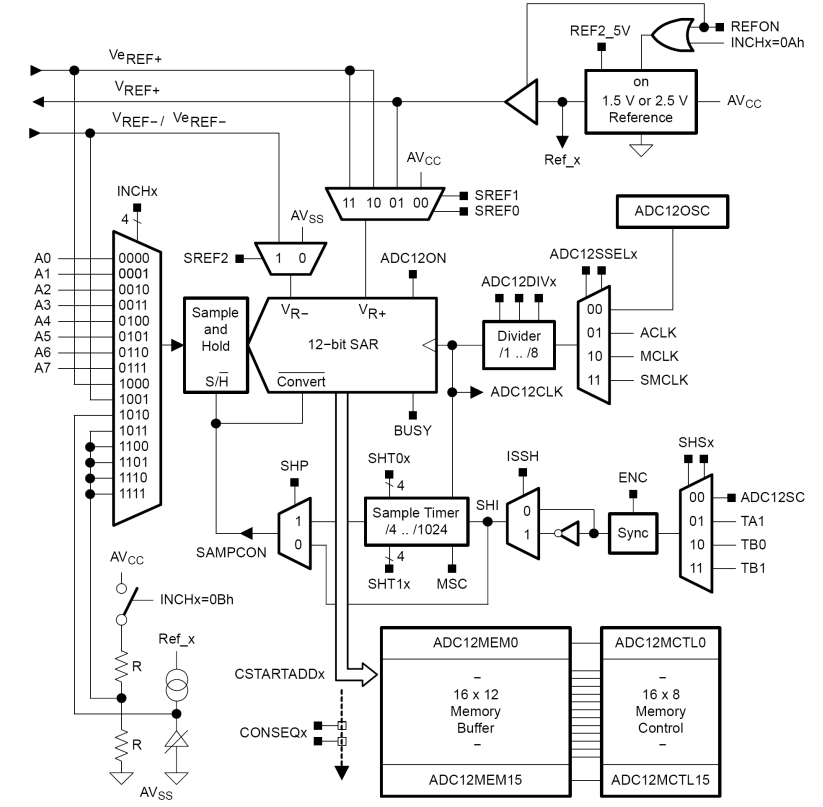
delay\_us(100);

}

发现采集的电平宽度有长，有短，107us是低电平宽度，207是高电平宽度

**ADC使用**

16通道里面有2个通道对参考电压采样



启动ADC方式

定时器启动，或者软件启动。这是定时启动通道

如果没有外接外部基准源，可以用内部基准源1.5V和2.5V可选

采样定时器

1.扩展采样模式

2.脉冲采样模式(我选)

时钟源分频器

也可以用ADC自身的时钟源，频率为5Mhz

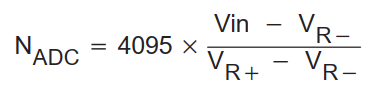
选择ADC时钟源

16通道里面有1个通道对内部温度传感器采样

16通道里面有4个通道对外部AVCC电压进行采样

ADC有16个转换通道，其中8个通道可以外接引脚

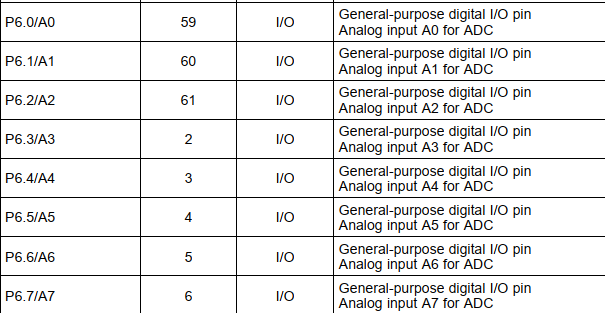
将ADC转换的电压数据存入缓冲器中



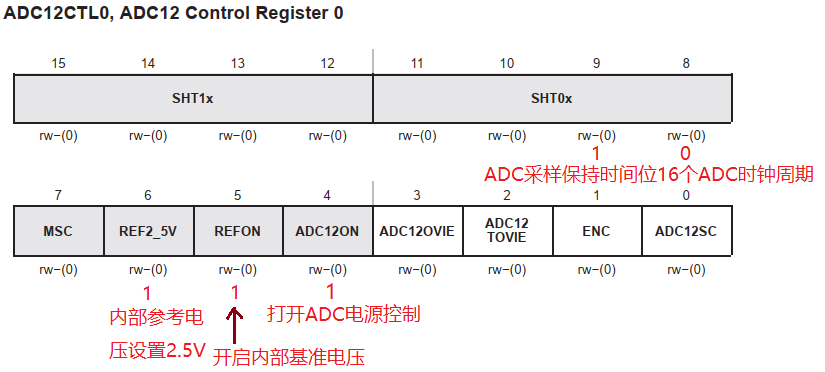
VR+:是基准源正极, VR-:是基准源负极，Vin是模拟通道输入电压。

如果基准源是正1.5V，那么VR- = 0V, VR+ = 1.5V, 如果Vin接地(0V),那么Nadc数字量为0

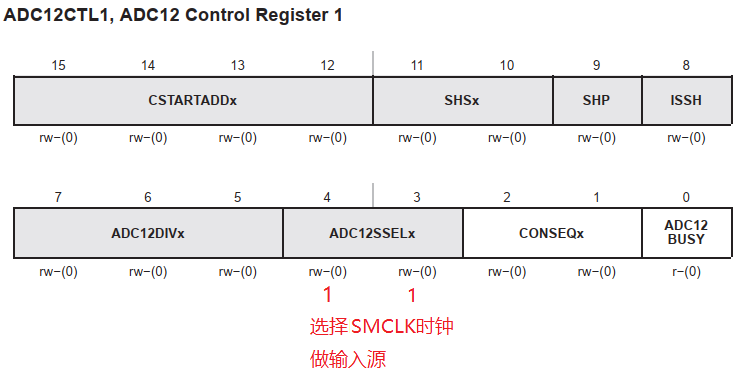
如果基准源是正1.5V，那么VR- = 0V, VR+ = 1.5V, 如果Vin=1.5V,那么Nadc数字量为4095(FFF)

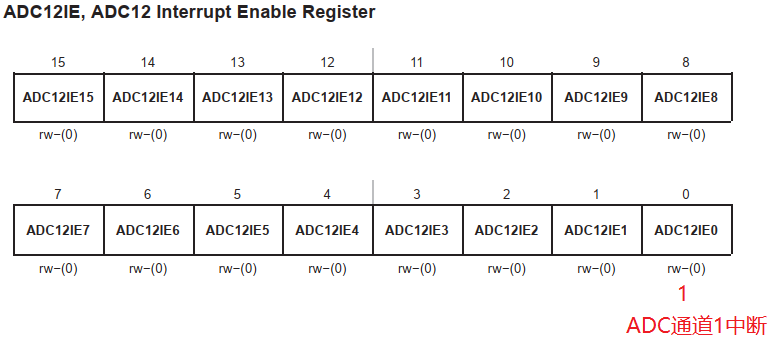


P6引脚是ADC 8个通道模拟输入引脚



采样保持时间越长，数据波动越小，数据值越稳定。采样保持时间越短，数据波动大，数据看起变动很大。





void ADC\_Init()

{

P6SEL|=0x01; //选择ADC通道0，P6.0 /A0功能模式

ADC12CTL0|= ADC12ON + SHT0\_2 + REF2\_5V + REFON; //ADC电源控制开，16个CLK，内部基准2.5V

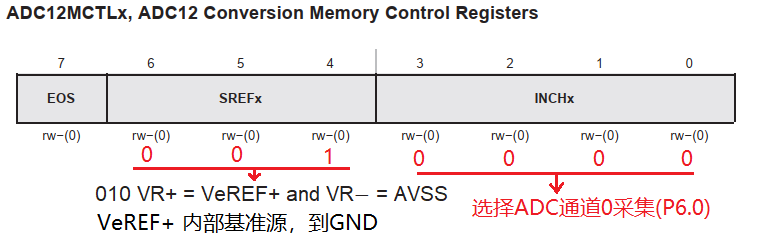
ADC12CTL1|= ADC12SSEL1 + ADC12SSEL0; //SMCLK做时钟源

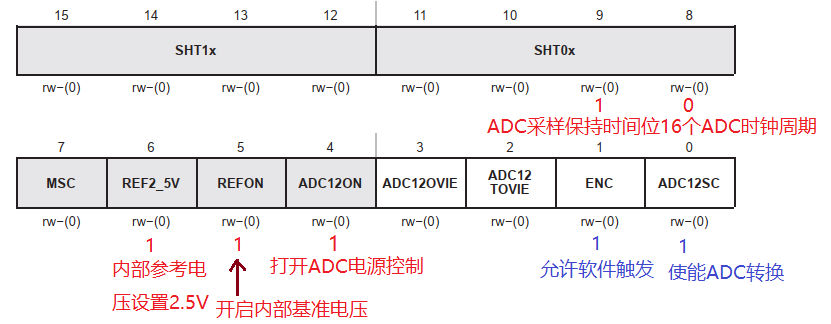
ADC12MCTL0= SREF0 + INCH\_0; //参考控制位及通道选择，这里选择通道0

ADC12IE|= 0x01; //中断允许 ADC通道1中断

ADC12CTL0|= ENC + ADC12SC; //使能转换器

}





灰色的这些寄存器一定是ENC设置为0时才能设置

单次ADC转换，使用P6.0/A0通道进行测试，

单次ADC转换适用于多路模拟开关采集多个模拟量，用模拟开关切换开关的方式送入同一个AD采集通道。

/\*\* ADC采集通道0数据

\*/

void ADC\_Init()

{

P6SEL|=0x01; //选择ADC通道0,P6.0/A0功能模式

ADC12CTL0|= ADC12ON + SHT0\_2 + REF2\_5V + REFON; //ADC电源控制开，16个CLK，内部基准2.5V

ADC12CTL1|= ADC12SSEL1 + ADC12SSEL0; //SMCLK做时钟源

ADC12MCTL0= SREF0 + INCH\_0; //参考控制位及通道选择，这里选择通道0

ADC12IE|= 0x01; //中断允许

ADC12CTL0|= ENC + ADC12SC; //使能转换器

}

unsigned int TEMP = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ADC中断服务程序

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#pragma vector=ADC\_VECTOR

\_\_interrupt void ADC12ISR(void)

{

while((ADC12CTL1&0x01)==1); //如果ADC忙，则等待，否则读取ADC转换数值

TEMP = ADC12MEM0; //读取ADC转换值

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

ADC\_Init();

\_EINT(); //开启总中断

printf("xxxzzzzz\n");

//开启转换

while(1)

{

ADC12CTL0 |= ADC12SC; //因为是单次ADC转换，所以每次转换要启动一次ADC，然后在中断中获取AD值

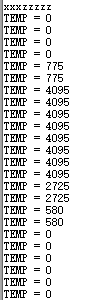
printf("TEMP = %d\n",TEMP);

delay\_ms(1000);

ADC12CTL0 &= ~ADC12SC; //因为是单次转换，必须转换之后要清0

}

}



过几秒才发现AD数据采集进来，这是因为中断触发了

开机发现ADC没采集到数据，这是因为用的延时，并不清楚ADC什么时候转换完成。所以最好在中断加标志位来处理

下面给ADC加入中断标志位

unsigned int TEMP = 0;

unsigned char ADCflag = 0; //给ADC加入中断标志位

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ADC中断服务程序

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#pragma vector=ADC\_VECTOR

\_\_interrupt void ADC12ISR(void)

{

while((ADC12CTL1&0x01)==1); //如果ADC忙，则等待，否则读取ADC转换数值

TEMP = ADC12MEM0; //读取ADC转换值

ADCflag = 1;

}

int main( void )

{

// Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

ADC\_Init();

\_EINT(); //开启总中断

printf("xxxzzzzz\n");

ADCflag = 1;//开启第1次ADC转换

while(1)

{

while(ADCflag == 1)

{

ADC12CTL0 |= ADC12SC; //因为是单次ADC转换，所以每次转换要启动一次ADC，然后在中断中获取AD值

ADC12CTL0 &= ~ADC12SC; //因为是单次转换，必须转换之后要清0

ADCflag = 0;

printf("TEMP = %d\n",TEMP);

delay\_ms(500);

//ADC12CTL0 |= ADC12SC; //因为是单次ADC转换，所以每次转换要启动一次ADC，然后在中断中获取AD值

}

}

}

使能SHP，使ADC转换结束后自动清0

unsigned int TEMP = 0;

unsigned char ADCflag = 0; //给ADC加入中断标志位

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ADC中断服务程序

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#pragma vector=ADC\_VECTOR

\_\_interrupt void ADC12ISR(void)

{

while((ADC12CTL1&0x01)==1); //如果ADC忙，则等待，否则读取ADC转换数值

TEMP = ADC12MEM0; //读取ADC转换值

ADCflag = 1;

}

/\*

\* ADC采集通道0数据

\*/

void ADC\_Init()

{

P6SEL|=0x01; //选择ADC通道0,P6.0/A0功能模式

ADC12CTL0|= ADC12ON + SHT0\_2 + REF2\_5V + REFON; //ADC电源控制开，16个CLK，内部基准2.5V

ADC12CTL1|= ADC12SSEL1 + ADC12SSEL0 + SHP; //SMCLK做时钟源，SHP AD转换结束后自动清0 ADC12SC，

ADC12MCTL0= SREF0 + INCH\_0; //参考控制位及通道选择，这里选择通道0

ADC12IE|= 0x01; //中断允许

ADC12CTL0|= ENC + ADC12SC; //使能转换器

}

int main( void )

{

WDTCTL = WDTPW + WDTHOLD; //关闭看门狗

Clock\_Init(); //初始化时钟

InitUSART(); //串口初始化

ADC\_Init();

\_EINT(); //开启总中断

printf("xxxzzzzz\n");

ADCflag = 1;//开启第1次ADC转换

while(1)

{

ADC12CTL0 |= ADC12SC; //因为是单次ADC转换，所以每次转换要启动一次ADC，然后在中断中获取AD值

while(ADCflag == 1)

{

// ADC12CTL0 &= ~ADC12SC; //加入了SHP 不需要手动清0

ADCflag = 0;

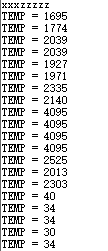
printf("TEMP = %d\n",TEMP);

delay\_ms(500);

}

}

}



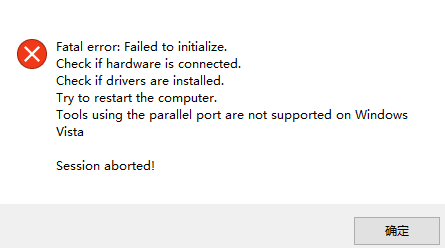
低电平

高电平采集正常

空气电压采集正常

ADC死循环查询法采集数据

IAR经过TI JTAG仿真器下载，如果某一天出现Fatal error: Failed to initialize. Check if hardware is connected. Check if drivers are installed. Try to restart the computer. Tools using the parallel port are not supported on Windows Vista Session aborted!



这不是操作问题，请把电脑的杀毒软件，电脑管家类的东西退出。就能正常下载了。

**MSP430 因为是16位单片机 函数返回值必须满足16位。如果返回值是32位数据类型，就会有问题 (重点)**

float Get()

{

float data = 100;

return data; //正常32位的单片机应该返回100，正常

}

int main()

{

float ret = 0;

ret = Get();

printf(“ret = %f\n”, ret); //我们发现ret得到的数据位0

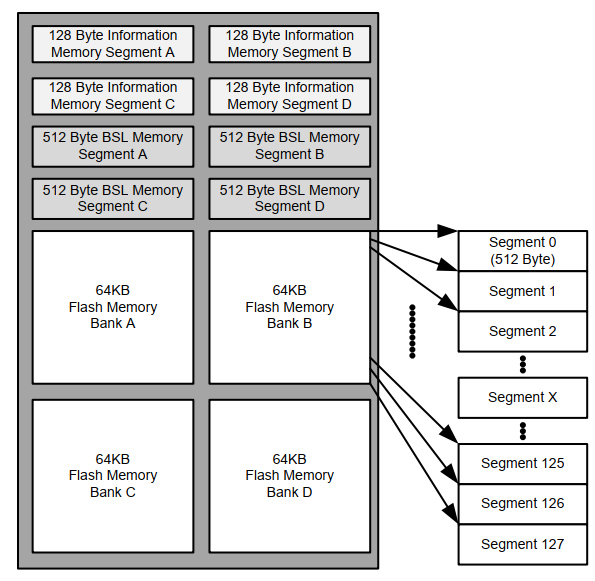
}

这就是16位单片机，函数返回值不能超过16位，所以只有返回int，或者char。

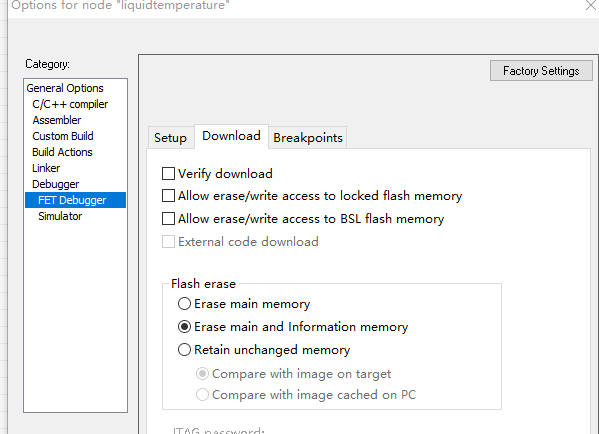
所以可以用指针做形参的方式，取地址，修改传入的变量。

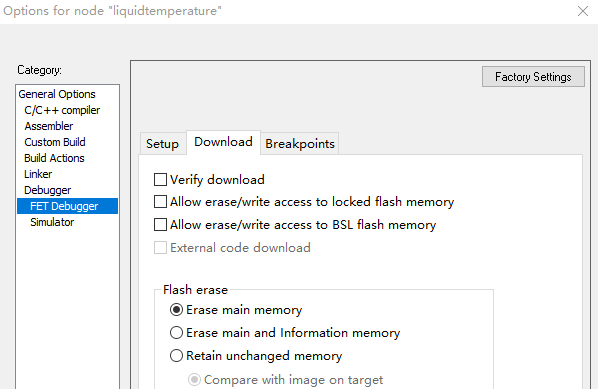
经过测试发现数据类型long做返回值没有问题。为什么double和float返回有问题呢?

**MSP430内部flash模拟EEPROM，如何防止已经存入的数据，在第二次用JTAG下载的时候不会被清空**



我们平时用flash做eeprom的区域是在Information区域

用IAR创建的工程，默认是下载程序的时候将flash中的main主程序区和information区域全部擦除掉



现在改成下载程序的时候只擦除main主程序区，这样你上次在information区域保存的设备数据就不会被擦除。做到eeprom永久保存功能。