我是 IAR 的新手，最近搞 avr 才使用的 IAR,在使用的过程中第一个遇到的问题就是很多寄存器不可以使用，比如 PORTC,PORTD 等等，这些明明是可以用的，不知道问什么不能用，后来问人终于知道了，这是 IAR 设置的问题，在设置里面，option->general options -> system -> enable bit defination in I/O - Include files 把位操作使能勾上就可以了.

==========

** IAR 嵌入式开发平台编译链接时产生 UBROF F8 格式即可导入到 AVR Studio 中调试。
** 按下面设置：Project -> Options -> LINKER -> Output -> Other -> UBROF F8
** 打开 AVR Studio，File -> Open File -> Debug\Exe*.d90 (有必要改变文件类型为“所有文件”)。

==========

IARAVR C 编译器的中断使用方法

IAR 公司开发的 AVR 单片机 C 编译器是一款非常优秀的开发工具，由于它的编译效率要比 ICCAVR、CODEVISIONAVR、GCCAVR 编译器都要高，很容易编写出高效的 C 程序。

对于 C 程序的编写，大体都是一样的，只是不同的编译器在标准 C 基础上都有自己的扩展特性。AVR 单片机的中断服务程序的编写对于不同的编译器声明的方法也就不同。例如在 IAR AVR C 编译器中使用定时器 Timer0 溢出中断声明的方法如下：

```
#pragma vector = TIMER0_OVF_vect
__interrupt void Timer0_OverFlow_Interrupt(void)
{
  //用户自己添加的程序段
}
```

"#pragma vector = "是必须声明的，等于号后面内容为对应中断向量地址

“__interrupt”这是 IAR AVR C 编译器中的中断服务程序声明的关键字，是必须的。后面就是用户给该中断服务程序取的函数名，由用户自己定义。

另外有一点必须注意的是：中断服务函数必须为无参，无返回参数的函数。

IAR 的中断使用和 GCC 有所区别：

例如：使用 ADC 中断

IAR 编译器：

```
#pragma vector = ADC_vect
__interrupt void adc(void)
```

GCC 编译器:SIGNAL(SIG_ADC)

IAR 可以直接进行调试，gcc 需要在 AVR Studio 中进行调试；

IAR 中定义了一些常用的内部函数在 intrinsics.h 和 comp_a90.h 等头文件中有函数声明；

IAR 中的寄存器位操作不能直接使用，需要宏定义，#define ENABLE_BIT_DEFINITIONS

IAR 的位操作功能强大，例如：PORTB_BIT2 = 1;

IAR AVR 中断应用
2009-08-08 14:48
如何输出 HEX 文件?
在配置文件后面加入以下代码,便可输出 HEX 文件,A90 文件与 HEX 文件一样,SLISP 都能识别.
// Output File
-Ointel-extended,(XDATA)=.eep //产生 eeprom 文件
-Ointel-extended,(CODE)=.A90 //产生烧写文件
-Ointel-extended,(CODE)=.hex //产生烧写文件

中断向量的使用
IAR 中定义中断函数的格式是
////////////////////////////////
#pragma vector=中断向量
__interrupt void 中断服务程序(void)
{
//中断处理程序
}
////////////////////////////////////
中断的初始化要另外加入代码，可在主程序内加入。如下是各个中断函数的定义。

```
//中断定义
#include <iom16.h>

#pragma vector=INT0_vect
__interrupt void INT0_Server(void)
{
}
#pragma vector=INT1_vect
__interrupt void INT1_Server(void)
{
}
#pragma vector=TIMER2_COMP_vect
__interrupt void TIMER2_COMP_Server(void)
{
}
#pragma vector=TIMER2_OVF_vect
```

```
__interrupt void TIMER2_OVF_Server(void)
{
}
#pragma vector=TIMER1_CAPT_vect
__interrupt void TIMER1_CAPT_Server(void)
{
}
#pragma vector=TIMER1_COMPA_vect
__interrupt void TIMER1_COMPA_Server(void)
{
}
#pragma vector=TIMER1_COMPB_vect
__interrupt void TIMER1_COMPB_Server(void)
{
}
#pragma vector=TIMER1_OVF_vect
__interrupt       void TIMER1_OVF_Server(void)
{
}
#pragma vector=TIMER0_OVF_vect
__interrupt       void TIMER0_OVF_Server(void)
{
}
#pragma vector=SPI_STC_vect
__interrupt       void SPI_STC_Server(void)
{
}
#pragma vector=USART_RXC_vect
__interrupt       void USART_RXC_Server(void)
{
}
#pragma vector=USART_UDRE_vect
__interrupt       void USART_UDRE_Server(void)
{
}
#pragma vector=USART_TXC_vect
__interrupt       void USART_TXC_Server(void)
{
}
#pragma vector=ADC_vect
__interrupt void ADC_Server(void)
{
}
#pragma vector=EE_RDY_vect
```

```
__interrupt void EE_RDY_Server(void)
{
}
#pragma vector=ANA_COMP_vect
__interrupt void ANA_COMP_Server(void)
{
}
#pragma vector=TWI_vect
__interrupt void TWI_Server(void)
{
}
#pragma vector=INT2_vect
__interrupt void INT2_Server(void)
{
}
#pragma vector=TIMER0_COMP_vect
__interrupt void TIMER0_COMP_Server(void)
{
}
#pragma vector=SPM_RDY_vect
__interrupt void SPM_RDY_Server(void)
{
}
```

如何把常数字符串定义在 flash 空间?

法一：unsigned char __flash temptab[] = {1,2,3,4,5};

法二：__flash unsigned char temptab[] = {1,2,3,4,5};

法三：#pragma type_attribute=__flash
unsigned char temptab[]={1,2,3,4,5};

法四：const unsigned char temptab[]={1,2,3,4,5};

注:第三种方式用#pragma 说明后，下面的定义的变量将都在 FLASH 空间了，用于定义一批 FLASH 变量,但实际上一般只能作为常
量使用了.

IAR For AVR 定时器中断初值计算方法

2009-08-10 23:57

使用芯片 AT Mega16 外部晶振 4.00MHz

定时器 1 （16 位定时器）寄存器 TCCR1B = 0x04 设定 256 预分频

要利用定时器定时 1 秒

1，4000000 / 256 = 15625　　说明定时器每当 1/15625 秒 就会触发一次中断

2，65535 - 15625 = 49910　　计算出要累加多少次才能在 1 秒后出发定时器 1 的溢出中断

3，49910 <==> C2 F6　　将计算后的值换算成 16 进制

4， TCNT1H = 0xC2 ;　　对寄存器赋值
TCNT1L = 0xF6 ;

==========

例如用 16 位定时器 TIMER1,4MHZ 晶振，256 分频，100ms 定时，如何求得初值赋给 TCNT1?

65536-(4M/256)*0.1=63973.5

其中，4M 是晶体频率，0.1 是定时时长单位秒。

对于 8 位的定时器
T=(2^8-计数初值)*晶振周期*分频数=(2^8-计数初值)/晶振频率*分频数
计数初值=2^8-T/晶振周期/分频数=2^8-T*晶振频率/分频数

因为 AVR 一指令 一周期

IAR FOR AVR 定时器中断的使用
2009-08-10 23:08

首先看下在 iar 里面 iom16.h 里面的中断向量表

/* NB! vectors are specified as byte addresses */

```
#define     RESET_vect          (0x00)
#define     INT0_vect           (0x04)
#define     INT1_vect           (0x08)
#define     TIMER2_COMP_vect    (0x0C)
#define     TIMER2_OVF_vect     (0x10)
#define     TIMER1_CAPT_vect    (0x14)
#define     TIMER1_COMPA_vect   (0x18)
#define     TIMER1_COMPB_vect   (0x1C)
```

```
#define     TIMER1_OVF_vect       (0x20)
#define     TIMER0_OVF_vect       (0x24)
#define     SPI_STC_vect          (0x28)
#define     USART_RXC_vect        (0x2C)
#define     USART_UDRE_vect       (0x30)
#define     USART_TXC_vect        (0x34)
#define     ADC_vect              (0x38)
#define     EE_RDY_vect           (0x3C)
#define     ANA_COMP_vect         (0x40)
#define     TWI_vect              (0x44)
#define     INT2_vect             (0x48)
#define     TIMER0_COMP_vect      (0x4C)
#define     SPM_RDY_vect          (0x50)
```

然后我是用的 atmega16　　4mhz 晶振

源程序：

```
#include <iom16.h>

char flag=0;

void timer_init()                    //    中断初始化
{
TCCR1B = 0x04;

TCNT1H = 0xc2;
TCNT1L = 0xf6;

TIMSK_Bit2 = 1;    //      定时器中断屏蔽寄存器
SREG_Bit7 = 1; //    总中断
}

# pragma vector = TIMER1_OVF_vect
__interrupt void timer1(void)
{
TCNT1H = 0xc2;
TCNT1L = 0xf6;
flag=1;
}

void main(void)
{
```

```
timer_init();
DDRB_Bit1 = 1;
while(1)
{
      if(flag==1)
      {
            PORTB_Bit1 = ~PORTB_Bit1;
            flag = 0;
      }
}
}
```

IAR For AVR USART 应用

2009-08-10 23:19

```
#include <iom16.h>
#define uchar unsigned char
#define uint unsigned int


//#####################################################################
/*串口初始化函数*/
void Uart_Init(void)
{
    UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE); //允许发送和接收
UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0); //8 位数据位+1 位停止位

UBRRH=0x00;                                                    //设置波特率寄存器低位
字节
          UBRRL=47;                                          //9600      //设置波特率寄存器高
位字节

DDRD_Bit1=1;                                        //配置 TX 为输出（很重要）
}
//#####################################################################
/*发送一个字符数据，查询方式*/
void Uart_Transmit(uchar data)
{
       while(!(UCSRA&(1<<UDRE)));
    //while(UCSRA_UDRE==0);     /*  等待发送缓冲器为空*/
UDR = data;                                  /*  发送数据*/
}
```

```
//#####################################################################
//发送一串数据 带回车符
void Uart_Puts(uchar *str)
{
while(*str)
            {
      Uart_Transmit(*(str++));
}
Uart_Transmit(0x0a);//回车换行
Uart_Transmit(0x0d);
}
//#####################################################################
//发送一串数据 不带回车符
void Uart_Put(uchar *str)
{
while(*str)
            {
      Uart_Transmit(*(str++));
}
}
//#####################################################################
/*数据接收，查询方式*/
unsigned char Uart_Receive( void ) {

while (!(UCSRA & (1<<RXC)));     /*  等待接收数据*/
return UDR;
}
//#####################################################################
```

IAR For AVR EEPROM 应用 2009-08-10 23:31 使用芯片 AT Mega16

```
#include<ina90.h>
```

__EEPUT(ADR,VAL)；//向指定 EEPROM 空间地址（ADR）中写入数据（VAL）

__EEGET(VAR,ADR)；//从指定 EEPROM 空间地址（ADR）中读取数据（VAL）

IAR For AVR 精确延时 2009-08-12 18:01C 语言中，想使用精确的延时程序并不容易。IAR 中有这样的一个函数 __delay_cycles(),该函数在头文件 intrinsics.h 中定义，函数的作用就是延

时 N 个指令周期。根据这个函数就可以实现精确的延时函数了（但不能做到 100%精确度）。

实现的方法：

建立一个 delay.h 的头文件：

```
#ifndef __IAR_DELAY_H
#define __IAR_DELAY_H

#include <intrinsics.h>

#define XTAL 8 //可定义为你所用的晶振频率（单位 Mhz）


#define delay_us(x) __delay_cycles ( (unsigned long)(x * XTAL) )
#define delay_ms(x) __delay_cycles ( (unsigned long)(x * XTAL*1000) )
#define delay_s(x) __delay_cycles ( (unsigned long)(x * XTAL*1000000) )

#endif
```

注意： __delay_cycles(x),x 必须是常量或则是常量表达式，如果是变量则编译报错！

IAR For AVR 串口中断接收 2009-08-14 18:20 应用芯片： AT Mega16 晶振：7.3728MHz

代码文件： uart_int.c

|_____DELAY.H

##

DELAY.H ：

```
#ifndef __IAR_DELAY_H
#define __IAR_DELAY_H

#include <intrinsics.h>

#define XTAL 7.3728                    //可定义为你所用的晶振频率（单位 Mhz）
```

```
#define delay_us(x) __delay_cycles ( (unsigned long)(x * XTAL) )
#define delay_ms(x) __delay_cycles ( (unsigned long)(x * XTAL*1000) )
#define delay_s(x) __delay_cycles ( (unsigned long)(x * XTAL*1000000) )

#endif
```

uart_int.c ：

```
#include <iom16.h>
#include "delay.h"
#define uchar unsigned char
#define uint unsigned int

uchar c;

//#####################################################################
/*串口初始化函数*/
void Uart_Init(void)
{
    UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);                 //允许发送和接收
    UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);          //8 位数据位+1 位停止位

    UBRRH=0x00;                                                                    //
设置波特率寄存器低位字节
    UBRRL=47;
//9600      //设置波特率寄存器高位字节

    SREG_I = 1;                                                                    //
开总中断
    DDRD_Bit1=1;                                                                   //
配置 TX 为输出（很重要）
}
//#####################################################################
/*发送一个字符数据，查询方式*/
void Uart_Transmit(uchar data)
{
     while(!(UCSRA&(1<<UDRE)));                                          // 等待发送缓冲
器为空

//也可以写成  while(UCSRA_UDRE==0);
     UDR                                =                                  data;
//  发送数据
}
```

```
//#################################################################
/*中断接收*/
#pragma vector=USART_RXC_vect
__interrupt void USART_RXC_Server(void)
{
UCSRB_RXCIE  =  0;                                                    //关串口
中断
c                                =                                UDR                                ;
//将收到的值赋值给变量
Uart_Transmit(c);                                                     //发给
串口以检测对错
UCSRB_RXCIE = 1;                                                      //开串口
中断
}
//#################################################################
/*主函数*/
void main(void)
{
Uart_Init();
delay_us(20);                          //串口初始化后，必须延时 20us 以上才能发送数据，
否则会出现错误
Uart_Transmit(0x64);

while(1)
{ ; }                                                 //此时可以用串口助手发送字符，然后可
以正确接收
}
```

IAR For AVR 两线串行接口

TWI 应用 2009-08-15 16:39

ATMEL 的 TWI 和 PHILIPS 的 IIC 基本上应该是算一个东西，但是他们在名义上是不同的，这样谁都不用支付给对方使用费。他们的协议是一样的，所有我们作为使用者基本可以简单的看成 TWI 就是 IIC 。

废话说完，开始正题。这次是关于在 ATMega16 平台下的硬件 IIC(还不太习惯说 TWI)的使用。在 ATMega16 的 Datasheet 里我们可以看到很强大的功能，主从设置很多。本文只说一种最常用的方式，那就是“ATMega16 硬件 TWI 的 扫描发送 和 扫描读取”。

首先要明确 TWI 发送和接受的流程：

发送：

1，设定数据传输波特率

2，发送 START 信号，等待应答 ==》 《== 应答信号

3，发送芯片地址，等待应答 ==》 《==应答信号

4，发送数据的绝对地址，等待应答 ==》 《==应答信号

5，发送要写入的数据，等待应答 ==》 《==应答信号

6，发送 STOP 信号，释放总线 ==》 数据写入成功

接收：

1，设定数据传输波特率

2，发送 START 信号，等待应答 ==》 《== 应答信号

3，发送芯片地址，等待应答 ==》 《==应答信号

4，发送数据的绝对地址，等待应答 ==》 《==应答信号

5，发送 RESTART 信号，等待应答 ==》 《==应答信号

6，发送芯片地址并注明读操作，等待应答 ==》 《==应答信号

7，读取数据，等待应答 ==》 《==应答信号

8，发送 STOP 信号，释放总线 ==》 数据读操作成功

应用芯片 ：ATMega 16 晶振 ： 7.3728

代码文件：Project

```
        |___TWI.C
        |          |______ IAR_DELAY.H
        |___UART.C
```

@@

IAR_DELAY.H

```
#ifndef __IAR_DELAY_H
#define __IAR_DELAY_H

#include <intrinsics.h>

#define XTAL 7.3728                        //可定义为你所用的晶振频率（单位 Mhz）


#define delay_us(x) __delay_cycles ( (unsigned long)(x * XTAL) )
#define delay_ms(x) __delay_cycles ( (unsigned long)(x * XTAL*1000) )
#define delay_s(x) __delay_cycles ( (unsigned long)(x * XTAL*1000000) )

#endif
```

@@

UART.C

```
#include <iom16.h>
#define uchar unsigned char
#define uint unsigned int

//##############################################################
/*串口初始化函数*/
void Uart_Init(void)
{
    UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);          //允许发送和接收
UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0); //8 位数据位+1 位停止位

UBRRH=0x00;                                                             //设置波特
率寄存器低位字节
UBRRL=47;                                                                   //9600
//设置波特率寄存器高位字节

DDRD_Bit1=1;                                                          //配置 TX 为
输出（很重要）
}
//##############################################################
/*发送一个字符数据，查询方式*/
void Uart_Transmit(uchar data)
{
      while(!(UCSRA&(1<<UDRE)));
    //while(UCSRA_UDRE==0);                                  /* 等待发送缓冲器为空*/
```

```
UDR = data;                                                  /*  发送数据*/
}


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

#include <iom16.h>
#include "IAR_DELAY.H"
#define uchar unsigned char
#define uint unsigned int

void Uart_Init(void);
void Uart_Transmit(uchar data);


//变量声明
#define EEPROM_BUS_ADDRESS 0xA0              //器件地址

/*#########################################################################*/
                                                                           /*
从器件地址位定义：________________________________________-------------*/

/*      AT24C02              | 1 | 0 | 1 | 0 | A2 | A1 | A0 | R/~W |------------*/
                                                                           /*
--------------------------------------------------*/

/*#########################################################################*/
//主机发送模式时各状态字的后续动作
#define TW_START                           0x08              //开始信号已发出
#define TW_REP_START                 0x10            //重复开始信号已发出
#define TW_MT_SLA_ACK                  0x18             //写字节已发出并受到 ACK 信号
#define TW_MT_SLA_NACK             0x20            //写字节已发出并受到 NACK 信号
#define TW_MT_DATA_ACK           0x28             //数据已发出并受到 ACK  信号
#define TW_MT_DATA_NACK         0x30          //数据已发出并受到 NACK  信号
#define TW_MT_ARB_LOST           0x38           //丢失总线控制权
//主机接收模式时各状态字的后续动作
#define TW_MR_ARB_LOST           0x38             //丢失总线控制权，未收到应答信号
#define TW_MR_SLA_ACK              0x40             //读命令已发出并受到 ACK
#define TW_MR_SLA_NACK             0x48             //读命令已发出并受到 NACK
#define TW_MR_DATA_ACK         0x50             //数据已收到，ACK 已发出
#define TW_MR_DATA_NACK         0x58             //数据已收到，NACK 已发出

#define IIC_Start()              TWCR =(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)        // TWINT 位  通
过写 1 进行清零，一旦清零则 TWI 开始工作，当相应硬件工作完成后  TWINT 位会重新置位
```

为 1

// TWSTA 位 会让硬件在总线上产生一个 START 的信号 ，声明自己希望成为主机

// TWEN 位 使能 TWI 功能,将 PC0 和 PC1 管脚切换到第二功能上来， 如果清零则为中断 TWI 的传输

#define IIC_Stop() TWCR =(1<<TWINT)|(1<<TWSTO)|(1<<TWEN) // TWSTO 位在主机模式下，会让硬件在总线上产生一个 STOP 得信号，并且 SCL 和 SDA 两个引脚位高阻态

#define IIC_Wait() while(!(TWCR&(1<<TWINT))) // TWINT 位 经过一次置位使硬件 TWI 开始工作 ，然后在检测 TWCR 寄存器的 TWINT 位是不是被置位, 如果置位为 1 则表示工作完成可以向下进行

```
//#############################################################################
#
/*I2C 总线单字节写入*/
unsigned char twi_write(unsigned char addr, unsigned char dd)
{
        TWBR = 10;                              //设定波特率

/*start 启动*/
IIC_Start();                                    //硬件发送 START 信号，并且清零 TWINT 位，使能硬件 TWI，使 TWI 开始工作
IIC_Wait();                                     //等待 发送 START 完成 TWINT 位置位
if ((TWSR & 0xF8) != 0x08) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量，如果正确则向下进行数据传输，错误返回 0

        /*SLA_W 芯片地址*/
TWDR = EEPROM_BUS_ADDRESS ;             //芯片地址 0xA0 ，赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN);    //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x18) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

        /*addr 操作地址*/
TWDR = addr;                                    //将写入数据的绝对地址 ，赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN);    //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x28) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，
```

如果正确则向下进行数据传输，错误返回 0

/*dd 写入数据*/
TWDR = dd; //将要写入的数据 ，赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN); //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait(); //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x28) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

/*stop 停止*/
IIC_Stop(); //数据传输完成，发送 STOP 信号，释放对总线的控制
return 1; //写入数据成功 ，返回 1 ，用来判断是否成功写入数据

}
//###
#
/*I2C 总线单字节读取*/
unsigned char twi_read(unsigned char addr)
{

unsigned char Receive_Byte ;
TWBR = 2; //设定波特率

/*start 启动*/
IIC_Start(); //硬件发送 START 信号，并且清零 TWINT 位，使能硬件 TWI，使 TWI 开始工作
IIC_Wait(); //等待 发送 START 完成 TWINT 位置位
if ((TWSR & 0xF8) != 0x08) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量，如果正确则向下进行数据传输，错误返回 0

/*SLA_W 芯片地址*/
TWDR = EEPROM_BUS_ADDRESS; //芯片地址 0xA0 ，赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN); //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait(); //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x18) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

/*addr 操作地址*/

```
TWDR = addr;                                    //将写入数据的绝对地址 ，赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN);    //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x28) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

        /*restart 重启动*/
IIC_Start();                                    //硬件发送 RESTART 信号，并且清零 TWINT 位，使能硬件 TWI，使 TWI 开始工作
        IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x10) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

       /*SLA_R 芯片地址*/
TWDR = 0xA1;                                    //芯片地址 0xA0 并注明是读取操作（最后一位为 1 ），赋值给数据寄存器 TWDR ，等待发送
TWCR = (1 << TWINT) | (1 << TWEN);    //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x40) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0

        /*读取数据*/
TWCR = (1 << TWINT) | (1 << TWEN);    //对控制寄存器 TWCR 的 TWINT 位软件写 1 进行清零，然后 使能 TWI 硬件接口 ，让 TWI 进行工作，发送 TWDR 寄存器 中的数据
IIC_Wait();                                     //等待数据发送完毕 TWINT 重新置位
if ((TWSR & 0xF8) != 0x58) return 0; //检测到 TWINT 位置位，比对 TWSR 寄存器内的状态量 ，如果正确则向下进行数据传输，错误返回 0
Receive_Byte = TWDR;                    //读取到的数据放到局部变量里

        /*stop 停止*/
IIC_Stop();                                     //数据传输完成，发送 STOP 信号，释放对总线的控制

        return Receive_Byte;                    //将读取到的数据作为函数的输出


}
//#############################################################################
#
/*主函数*/
void main(void)
```

```
{
uchar c,d;
Uart_Init();                                    //串口初始化
delay_us(20);
Uart_Transmit(0x55);                            //测试串口

c = twi_write(0x51,0xf8);                      //在地址 0x51 里写入数据 0x22
Uart_Transmit(c);                               //将返回值发送到串口测试是否写入成功

delay_ms(2);

d = twi_read(0x51);                             //将地址 0x51 里的数据读出来
Uart_Transmit(d);                               //将读取到的数据发送串口
while(1);
}
```

IAR AVR WatchDog 使用 2009-08-29 23:44 首先要包括 comp_a90.h 头文件 ， 他里面有喂狗程序 _WDR() 。

```
#include <iom16.h>
#include <intrinsics.h>
#include <comp_a90.h>
#include "delay.h"


void watchdog_init(void)
{
WDTCR |= ((1 << WDTOE) | (1 << WDE));              /*启动时序*/
WDTCR = ((1<< WDE) | (1 << WDP2) | (1 <<WDP1)); /*设定周期为 1S*/
}

void main(void)
{
watchdog_init();
DDRB_Bit0 = 1;

PORTB_Bit0 = 0 ;
delay_ms(500);
_WDR();

PORTB_Bit0 = 1 ;
delay_ms(500);
_WDR();
```

```
while(1){;}
```