七 I2C接口

这一节，我们将学习STM8S的I2C总线，I²C强大用途在于微控制器的应用，利用两根通用的输入输出接脚及软件的规划，可以让STM8S控制一个小型网络。一些常见的应用如下：

-为了保存系统的设置而存取EEPROM芯片。
-存取低速的数模转换器（DAC）
-存取低速的模数转换器（ADC）
-改变音量大小。
-取得硬件监视及诊断资料，例如中央处理器的温度及风扇转速
-读取实时的时钟（Real-time clock）
-在系统设备中用来开启或关闭电源供应

按照前几节的惯例，我们将使用一个小实验来完成I2C总线的学习。本节实验就是：读取和写入开发板上的AT24C64的指定位置的字节数据。

STM8S的I2C总线概述：

-多主机功能：该模块既可做主设备也可做从设备
-I2C主设备功能
　产生时钟
　产生起始和停止信号
-I2C从设备功能
　可编程的 I2C地址检测
　停止位检测
-产生和检测7位/10位地址和广播呼叫
-支持不同的通讯速度
　标准速度(最高 100 kHz)
　快速(最高 400 kHz)
-状态标志：
　发送器/接收器模式标志
　字节发送结束标志
　I2C总线忙标志

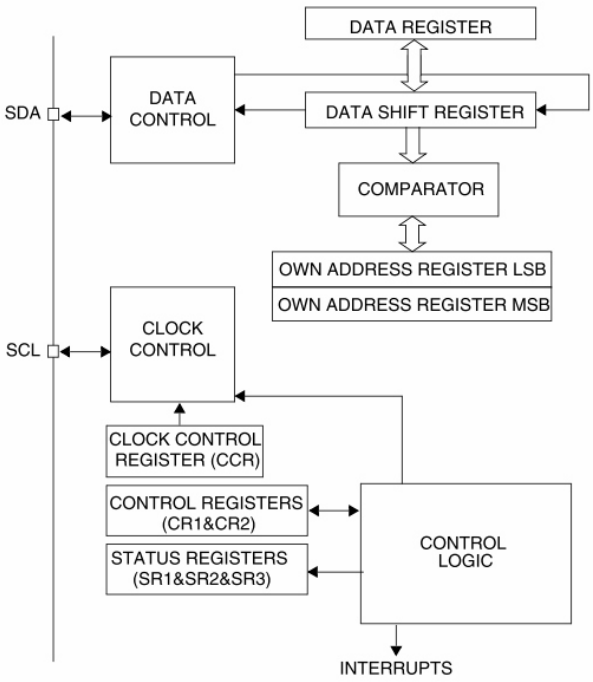
STM8S的I2C模式选择

接口可以下述4种模式中的一种运行：

- 从设备发送模式
- 从设备接收模式
- 主设备发送模式
- 主设备接收模式

注意

默认条件下，I2C模块工作于从模式。接口在产生起始条件后自动地从从模式切换到主模式；当仲裁失败或发送STOP信号时，则从主模式切换到从模式。允许多主机功能。



I2C的功能框图

需要做什么:

首先，当读写AT24C64时，I2C是工作在主模式下。通过查阅资料得知要想让STM8S的I2C进入主控制模式，以下是主模式所要求的操作顺序：

- 在I2C_FREQR寄存器中设定该模块的输入时钟以产生正确的时序
- I2C模块的输入时钟频率必须至少是:
  标准模式下为：1MHz
  快速模式下为：4MHz
- 配置时钟控制寄存器
- 配置上升时间寄存器
- 编程I2C_CR1寄存器启动I2C模块

按照以上的操作顺序，我们先来认识一下I2C的相关寄存器

控制寄存器(I2C-CR1)

7	6	5	4	3	2	1	0
NOSTRETCH	ENGC	保留					PE
rw	rw						rw

位7	**NOSTRETCH**：时钟延展禁止(从模式) 该位用在从模式下当ADDR或者BTF标志置位时，是否禁止时钟延展，直到软件将该位复位。 0：时钟延展使能； 1：时钟延展禁止。
位6	**ENGC**：广播呼叫使能 0：广播呼叫禁止，对地址00h不响应； 1：广播呼叫使能，对地址00h响应。
位5：1	保留，读出0
位0	**PE**：I²C模块使能 0：禁用I²C模块； 1：启用I²C模块：相应的I/O口需配置为复用功能。 **注：**如果清除该位时通讯正在进行，在当前通讯结束后，I²C模块被禁用并返回空闲状态。 由于PE=0，通讯结束后所有的位被置为0。

I2C-CR1中，我们需要设置I2C使能。即：I2C->CR1=0x01;

控制寄存器 2(I2C_CR2)

7	6	5	4	3	2	1	0
SWRST	保留			POS	ACK	STOP	START
rw				rw	rw	rw	rw

位7	**SWRST**：软件复位 当该位置为1，I2C模块处于复位状态。确保I2C总线被释放，并且总线空闲，然后再复位该位 0：I2C模块不在复位状态； 1：I2C模块处于复位状态。 **注意**：可以用于当STOP没有被检测到，而使得BUSY位一直置为1的情况
位6:4	保留位，读为0。

位3	**POS**：应答的位置(接收数据时) 该位可以被软件置位或者清零；也可以当PE=0时被硬件清零。 0：ACK位控制被移位寄存器正在接收的这个当前字节的应答或者不应答； 1：ACK位控制下一个将被移位寄存器接收的字节的应答或者不应答 注意：该位必须在数据接收开始前配置
位2	**ACK**：应答使能 该位可以被软件置位或者清零；也可以当PE=0时被硬件清零。 0：不返回应答； 1：收到一个字节后(匹配的地址字节或者数据字节)后返回应答。
位1	**STOP**：停止位产生 该位可以被软件置位或者清零；硬件也可以测到停止位后将该位清零；当超时错位被检测到时，硬件也会将该位置为1。 **主模式：** 0：不产生停止位 1：当前字节传输完成后，或者当前起始位发送完后，产生停止位。 注意：发送停止位前，必须清除I2C_SR1寄存器中的BTF位。 **从模式：** 0：没有停止位 1：当前字节传输完后，释放SCL和SDA线。
位0	**START**：起始位产生 该位可以被软件置位或者清零；也可以当PE=0时被硬件清零，或者起始位发送完后由硬件清零。 **主模式：** 0：不产生起始位 1：产生重复起始位 **从模式：** 0：不产生起始位 1：当总线空闲时产生起始位

I2C_CR2寄存器在数据传输的时候很重要。比如，设置应答位使I2C 在接收完一个字节数据后产生ACK；或者是在传送的开始发送一个开始位信号;在结束传送时，发送一个停止位。

这个寄存器我将不进行初始化设置。在传输过程中再进行相应的设置。

```
I2C->CR2=0x00;
```

频率寄存器(I2C_FREQR)

7	6	5	4	3	2	1	0
保留		FREQ[5:0]					
r	r	rw	rw	rw	rw	rw	rw

位7:6	保留位，读为0。
位5:0	**FREQ[5:0]**：外设时钟频率(1) 为了产生正确的时序，必须配置合适的输入时钟频率： 允许的时钟范围在1MHz和50MHz之间 000000：不允许 000001：1MHz 000010：2MHz … 110010：50MHz 不允许更高的值

1. I2C总线时序要求最小的外设时钟频率为：标准模式1MHz；快速模式4MHz

在这个应用中，我们需要访问AT24C64存储芯片。在时钟速度上选择400K。按照之前的设置说明，必须把外设时钟频率设置成10M 也就是：

```
I2C->FREQR =0x0A;
```

那如何设置I2C的时钟为400K呢？且看时钟控制寄存器。

时钟控制寄存器高位部分(I2C_CCRH)

F/S	DUTY	保留	CCR[11:8]
rw	rw	r	rw

位7	**F/S：**I2C主模式选择 0：标准模式I2C 1：快速模式I2C
位6	**DUTY：**快速模式下的占空比 0：快速模式 t_{low}/t_{high} = 2; 1：快速模式t_{low} /t_{high} = 16/9(参见CCR)
位5：4	保留，必须为0
位3：0	**CCR[11:8]：**时钟控制寄存器(主模式) 控制主模式下的SCLH时钟。 在I²C标准模式下： t_{high} = CCR × t_{CK} t_{low} = CCR × t_{CK} 在I²C快速模式下： 如果DUTY = 0： t_{high} = CCR × t_{CK} t_{low} = 2 × CCR × t_{CK} 如果DUTY = 1：(速度达到400kHz) T_{high} = 9 × CCR × t_{CK} T_{low} = 16 × CCR × t_{CK} 比如：标准模式下，为了产生100KHz的SCL频率： 假设FREQR = 08, t_{CK} = 125ns，那么CCR就必须为28h (28h ← → 40d *125 ns = 5000ns) **注意：**1. t_{high}包含SCL的上升边沿； 2. t_{low}包含SCL的下降边沿； 3. 这些时序均没有过滤器。

低位部分(I2C_CCRL)

7	6	5	4	3	2	1	0
CCR[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

位7:0	**CCR[7:0]：**时钟控制寄存器(主模式) 控制主模式下的SCLH时钟。 在I²C标准模式下： t_{high} = CCR × t_{CK} t_{low} = CCR × t_{CK} 在I²C快速模式下： 如果DUTY = 0： t_{high} = CCR × t_{CK} t_{low} = 2 × CCR × t_{CK} 如果DUTY = 1：(速度达到400kHz) T_{high} = 9 × CCR × t_{CK} T_{low} = 16 × CCR × t_{CK}

注意：

1 只有I2C禁止时(PE=0)才能配置CCR寄存器

2 要产生快速时钟400KHz，fCK 要是10MHz的整数倍

3 要产生100KHz的标准时钟，要求fCK ≥ 1MHz

所以要想得到400K的时钟速度，快速模式下，为了产生400KHz的SCL频率：FREQR = 10M，即tCK = 100ns，那么CCR就必须为 8（高电平800ns,低电平1600ns 周期2400ns.接近400kHz时钟.）：

```
I2C->FREQR =0x0A;  //10M
I2C->CCRH =0x80; //时钟高电平和低电平时间比 1:2
I2C->CCRL =8;
```

通过以上的设置，I2C就已经有了时钟信号，工作起来了。但是还不能满足我们的应用，我们需要发送和接收数据。

数据寄存器(I2C_DR)

DR[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw

位7:0	**DR[7:0]**：数据寄存器(1)(2)(3) 用于存放接收到的数据或放置用于发送到总线的数据 发送模式：当写一个字节至DR寄存器时，自动启动数据传输。一旦传输开始(TxE=1)，如果能及时把下一个需传输的数据写入DR寄存器，I²C模块将保持连续的数据流。 接收模式：接收到的字节被拷贝到DR寄存器(RxNE=1)。

I2C的数据寄存器是收发一体的，当需要传输字节时，将要发送到数据与到I2C_DR，模块会自动把这个数据发送出去。判断RxNE是否为1，来读取I2C_DR得到I2C接收到的数据.

状态寄存器1(I2C_SR1)

7	6	5	4	3	2	1	0
TxE	RxNE	保留	STOPF	ADD10	BTF	ADDR	SB
r	r	r	r	r	r	r	r

位7	**TxE**: 数据寄存器为空(发送时)(1) 0：数据寄存器非空； 1：数据寄存器空。 – 在发送数据时，数据寄存器为空时该位被置'1'，在发送地址阶段不设置该位。 – 软件写数据到DR寄存器可清除该位；或在发生一个起始或停止条件后，或当PE=0时由硬件自动清除。
位6	**RxNE**: 数据寄存器非空(接收时)(2)(3) 0：数据寄存器为空； 1：数据寄存器非空。 – 在接收时，当数据寄存器不为空，该位被置'1'。在接收地址阶段，该位不被置位。 – 软件对数据寄存器的读写操作清除该位，或当PE=0时由硬件清除。
位5	保留位，硬件强制为0
位4	**STOPF**: 停止条件检测位(从模式)(4) 0：没有检测到停止条件； 1：检测到停止条件。 – 在一个应答之后(如果ACK=1)，当从设备在总线上检测到停止条件时，硬件将该位置'1'。 – 软件读取SR1寄存器后，对CR2寄存器的写操作将清除该位，或当PE=0时，硬件清除该位。

位3	**ADD10**: 10位头序列已发送(主模式)(5) 0：没有ADD10事件发生； 1：主设备已经将第一个地址字节发送出去。 – 在10位地址模式下，当主设备已经将第一个字节发送出去时，硬件将该位置'1'。 – 软件读取SR1寄存器，接着将第二个地址字节写入DR，可以清除该位；或当PE=0时，硬件清除该位。
位2	**BTF**：字节发送结束(6)(7) 0：数据字节发送未完成； 1：数据字节发送结束。 –当NOSTRETCH=0时，在下列情况下硬件将该位置'1'： – 在接收时，当收到一个新字节(包括ACK脉冲)且数据寄存器还未被读取(RxNE=1)。 – 在发送时，当一个新数据将被发送且数据寄存器还未被写入新的数据(TxE=1)。 – 在软件读取SR1寄存器后，对数据寄存器的读或写操作将清除该位；或在传输中发送一个起始或停止条件后，或当PE=0时，由硬件清除该位。
位1	**ADDR**：地址已被发送(主模式)/地址匹配(从模式)(7) 在软件读取SR1寄存器后，对SR3寄存器的读操作将清除该位，或当PE=0时，由硬件清除该位。 **地址匹配(从模式)** 0：地址不匹配或没有收到地址； 1：收到的地址匹配。 – 当收到的从地址与OAR寄存器中的内容相匹配、或发生广播呼叫时(当对应的设置被使能时)，硬件就将该位置'1'。 **地址已被发送(主模式)** 0：地址发送没有结束； 1：地址发送结束。 – 10位地址模式时，当收到地址的第二个字节的ACK后该位被置'1'。 – 7位地址模式时，当收到地址的ACK后该位被置'1'。 **注：**在收到NACK后，ADDR位不会被置位。
位0	**SB**：起始位(主模式)(7) 0：未发送起始条件； 1：起始条件已发送。 – 当发送出起始条件时该位被置'1'。 – 软件读取SR1寄存器后，写数据寄存器的操作将清除该位，或当PE=0时，硬件清除该位。

状态寄存器2（I2C_SR2）

保留	WUFH	保留	OVR	AF	ARLO	BEER
	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0

位7：6	保留位，读出0
位5	**WUFH**：从停机(Halt)模式唤醒 0：没有从Halt模式唤醒； 1：Halt模式下，7位地址或者地址头序列匹配；或者是主模式下进入Halt模式。 **注意**：该位在从模式下(Halt模式)的被置位是异步的。只有ITEVTEN=1时才会被置位。 -- 软件写0，清除此位；或者PE=0时由硬件清零。
位4	保留位，读出0

位3	**OVR**：上溢/下溢 0：没有发生上溢/下溢； 1：发生了上溢/下溢。 -- 当NOSTRETCH=1，从模式，并且满足以下条件，由硬件置位： -- 接收时：当DR寄存器中的内容还没有读出，又收到新的字节(包括ACK脉冲)。新收到的字节因此被丢失。 --发送时：该发新的数据了，DR寄存器还没有被写入数据。同样的字节将会被发送两次。 软件写0可以清除该位；或者当PE=0时由硬件清零。 **注意**：如果写DR的时刻非常接近于SCL的上升沿，发送的数据是不能确定的，并且会产生保持时间错误。
位2	**AF**：应答失败 0：没有应答失败； 1：应答失败。 当没有返回应答时，由硬件置为1。 软件写0清除该位；或者当PE=0时由硬件清零。
位1	**ARLO**：仲裁失败(主模式) 0：没有检测到仲裁失败； 1：检测到仲裁失败。 当该模块丢失了对总线的仲裁控制并转交给其他主设备，硬件自动置位。 软件写0清除该位；或者当PE=0时由硬件清零。 仲裁失败发生后，模块自动切换回从模式(M/SL=0)
位0	**BERR**：总线错误 0：正常的起始或者结束条件； 1：错误的起始或者结束条件。 当硬件检测到错误的起始或者结束条件后，自动置为1； 软件写0清除该位；或者当PE=0时由硬件清零。

状态寄存器3（I2C_SR3）

7	6	5	4	3	2	1	0
保留			GENCALL	保留	TRA	BUSY	MSL
r	r	r	r	r	r	r	r

位7：5	保留位，读出0
位4	**GENCALL**：广播呼叫头序列(从模式) 0：没有广播呼叫； 1：当ENGC=1时收到了广播呼叫地址头序列。 -- 总线上出现结束或者重复起始条件后，或者在PE=0时，由硬件清零。
位3	保留位，读出0
位2	**TRA**：发送器/接收器 0：接收数据； 1：发送数据。 该位在整个寻址阶段结束时，根据地址字节的R/W位来决定。 当检测到结束条件(STOPF=1)，重复起始条件，总线仲裁失败(ARLO=1)，或者PE=0时，由硬件清零。

状态寄存器3（I2C_SR3）

位1	**BUSY**：总线忙 0：总线上没有通信； 1：总线上有通信。 -- 硬件检测到SDA或者SCL变成低电平，该位置位； -- 检测到结束条件时，硬件清零该位。 该位表明总线上时候正有通信在进行。即使模块没有使能的情况下(PE=0)，该位也有效。
位0	**MSL**：主/从模式 0：从模式； 1：主模式。 -- 当模块处于主模式(SB=1)，硬件置位； -- 检测到总线上出现结束条件，或仲裁失败，或者当PE=0时，由硬件清零。

中断寄存器（I2C_ITR）

保留					ITBUFEN	ITEVTEN	ITERREN
r	r	r	rw	rw	rw	rw	rw

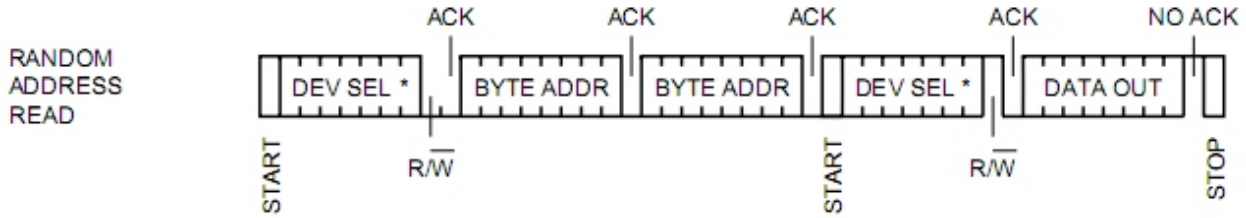
位7：3	保留位，读出0
位2	**ITBUFEN**：缓冲中断使能 0：TxE=1或者RxNE=1不产生任何中断； 1：TxE=1或者RxNE=1产生事件中断。
位1	**ITEVTEN**：事件中断使能 0：事件中断禁止； 1：事件中断使能。 发生以下事件时，产生中断： -- SB=1(主模式) -- ADDR=1(主/从模式) -- ADD10=1(主模式) -- STOPF=1(从模式) -- BTF=1，而没有TxE或者RxNE中断 -- TxE事件，如果ITBUFEN=1 -- RxNE事件，如果ITBUFEN=1 -- WUFH=1(从halt模式唤醒的异步中断)
位0	**ITERREN**：错误中断使能 0：错误中断禁止； 1：错误中断使能。 当发生以下错误时，产生中断： -- BERR=1； -- ARLO=1； -- AF=1； -- OVR=1。

在这个应用中，只是读写24C64，将使用查询的方式进行通信。不使用中断方式。即

```
I2C->ITR=0x00;
```

如何读取AT24C64的数据？

经过以上的设置STM8S的I2C已以准备好，可以接受你的命令。但是，我们如何读取AT24C64 内部指定位置的字节数据呢？且看它的操作手册里面的读时序：



要读取指定位置ADDR的数据，需要有以下过程：

1. 发送I2C始起信号
2. 发送AT24C64的器件地址 0xA0+写操作
3. 发送DDR的高8位地址
4. 发送DDR的低8位地址
5. 再发送I2C始起信号
6. 发送AT24C64的器件地址 0xA0+读操作
7. AT24C64返回数据
8. STM8S发送停止信号。读操作结束

回到问题的关键，我们如何控制STM8S发送这些信号呢？

以读取0001地址的数据为例：

1. 产生I2C始起信号

```
I2C->CR2 |= 0x01;  //产生始起信号
While((I2C->SR1&0x01)==0); //等待始起信号已发送
```

2. 发送AT24C64的器件地址 0xA0+写操作

```
I2C->DR = 0xA0;
While((I2C->SR1&0x02)==0); //等待接收到从AT24C64返回的ACK
Temp = I2C->SR3; //读SR3寄存器，以清除标志位ADDR
```

3. 发送DDR的高8位地址

```
I2C->DR = 0x00;
While((I2C->SR1&0x80)==0); //等待数据寄存器空
```

4. 发送DDR的低8位地址

```
I2C->DR = 0x01;
While((I2C->SR1&0x80)==0); //等待数据寄存器空
```

5. 再发送I2C始起信号　同步骤1

6. 发送AT24C64的器件地址 0xA0+读操作 I2C->DR = 0xA1; 同步骤2

7. AT24C64返回数据

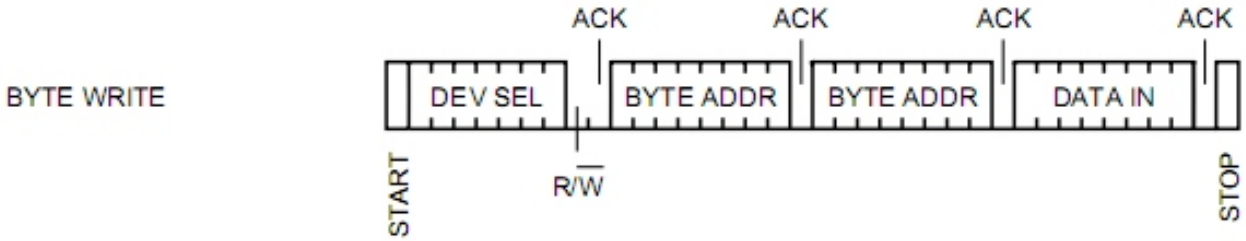
```
While((I2C->SR1&0x40)==0); //等待数据寄存器不为空
READ=I2C->DR ;
```

8. STM8S发送停止信号。读操作结束

```
I2C->CR2 |= 0x02;  //产生停止信号
```

如何向AT24C64写入指定数据？

通过上面的步骤，我们可以读取AT24C64内部的数据，下面我们来学习如何写入数据，且看操作手册里面的写时序：



要写入指定数据到地址DDR，需要有以下过程：

1. 发送I2C始起信号
2. 发送AT24C64的器件地址 0xA0+写操作
3. 发送DDR的高8位地址
4. 发送DDR的低8位地址
5. 发送要写的数据
8. STM8S发送停止信号。写操作结束

同样我们以写0xA5 到0002地址为例：

1. 产生I2C始起信号

```
I2C->CR2 |= 0x01;  //产生始起信号
While((I2C->SR1&0x01)==0); //等待始起信号已发送
```

2. 发送AT24C64的器件地址 0xA0+写操作

```
I2C->DR = 0xA0;
While((I2C->SR1&0x02)==0); //等待接收到从AT24C64返回的ACK
Temp = I2C->SR3; //读SR3寄存器，以清除标志位ADDR
```

3. 发送DDR的高8位地址

```
I2C->DR = 0x00;
While((I2C->SR1&0x80)==0); //等待数据寄存器空
```

4. 发送DDR的低8位地址

```
I2C->DR = 0x02;
While((I2C->SR1&0x80)==0); //等待数据寄存器空
```

5. 写入数据

```
I2C->DR = 0xA5;
While((I2C->SR1&0x80)==0); //等待数据寄存器空
```

6. STM8S发送停止信号。写操作结束

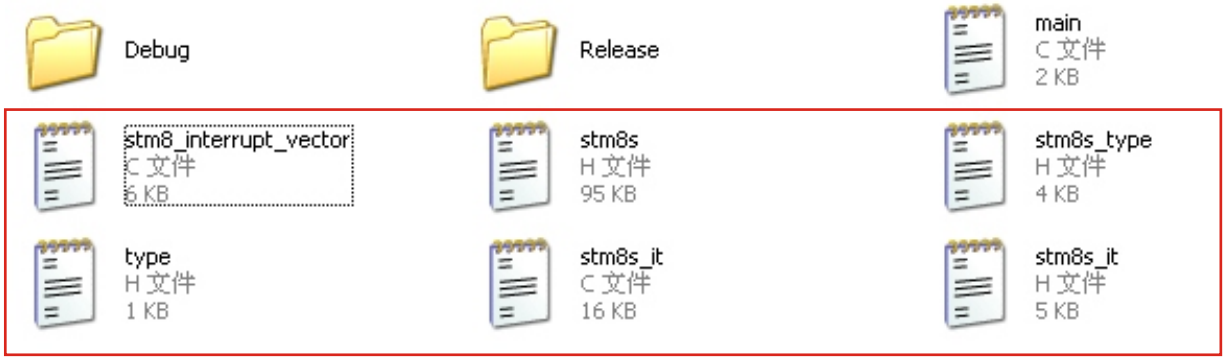
```
I2C->CR2 |= 0x02;  //产生停止信号
```

至此，我们已经完成了读写I2C的理论分析。接下来，我们将动手编写代码。

创建工程:

我们实现的功能是将指定的数据写入AT24C64，再读出来，比较看读出的是否和写入的一致。

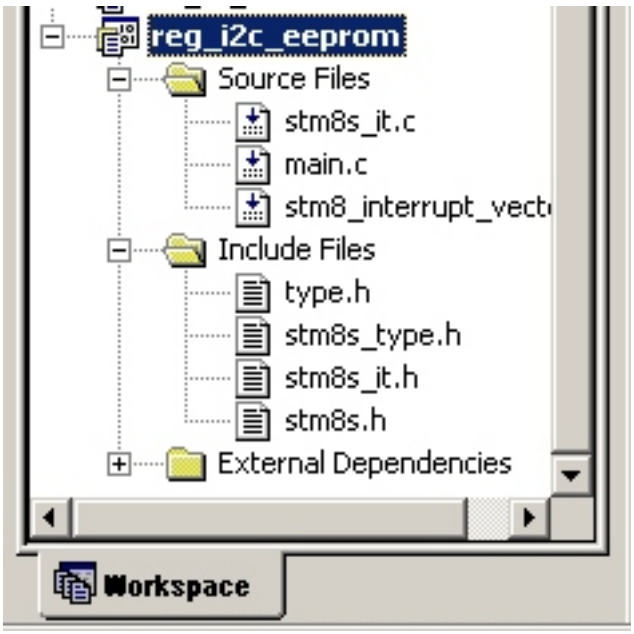
OK,那我们开始吧。
打开ST Visual Develop ，新建一个工程[Reg_I2C_EEprom]。选择目标芯片为STM8S208R8,然后把直接操作寄存器的模版工程文件复制一份过来。如下图：



把这6个必须的文件复制到你的工程目录下面。

stm8_interrupt_vector.c 中断向量表
stm8s.h STM8S所有的外设的定义，有了它，你才能在程序中操作寄存器
stm8s_type.h type.h 数据类型定义
stm8s_it.c 中断子程序。

把这些文件添加到工程中，源文件.c添加到工程目录的Source Files文件夹中，头文件.h添加到Include Files目录中。最终如下图：



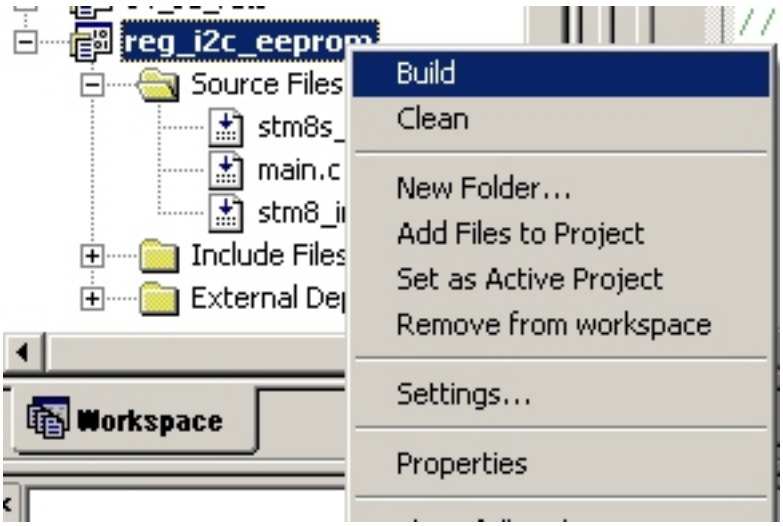
编写代码:

打开main.c 编写以下代码：

```
/* MAIN.C file
STM8S-EK 开发板相关例程
编写者：lisn3188
网址：www.chiplab7.com
作者E-mail：lisn3188@163.com
编译环境：ST Visual Develop + STM8 Cosmic
初版时间: 2011-12-18
测试： 本程序已在第七实验室的STM8S-EK上完成测试，实现功能见主程序main
 */
 #include "stm8s.h"

#define AT24C64_Chip  0xA0
/*
功    能:       往AT24C64的0x0023地址写入0xA5 然后再从这个地址读出，
     拿读到的和写入的数据比较，如果相等，两个LED亮，否则，只有LED1亮*/
main()
{
     unsigned char Temp,Byte_Write=0xA5,Byte_Read;
     unsigned int Addr=0x0023,i;
     GPIOC->DDR |= (u8)0x06;
     GPIOC->CR1 |= (u8)0x06;
     GPIOC->CR2 &= ~(u8)0x06;
     GPIOC->ODR |= (u8)0x02; //关灯
     GPIOC->ODR |= (u8)0x04;
//--------I2C模块初始化----------------------------------
     I2C->CR1 = 0x00;  //禁用I2C模块 以初始化I2C模块
     I2C->CR2 = 0x04;  //应答使能
     I2C->FREQR = 10;  //输入时钟频率 10M
     I2C->OARH  = 0x40; //设置地址模式配置 为1
     I2C->OARL  = 0x00; //做为从机时的7位地址
     I2C->ITR  =  0x00;  //禁止所有的I2C中断
     I2C->CCRH = 0x80; //快速模式I2C  快速模式 tlow/thigh = 2;
     I2C->CCRL = 8;  //400Khz 时钟速率
     I2C->CR1 = 0x01;  //启用I2C模块
//--------I2C模块初始化完成------------------------------
     I2C->CR2 |= 0x01;  //产生始起信号
     while((I2C->SR1&0x01)==0); //等待始起信号已发送
     I2C->DR = AT24C64_Chip;
     while((I2C->SR1&0x02)==0); //等待接收到从AT24C64返回的ACK
     Temp = I2C->SR3; //读SR3寄存器，以清除标志位ADDR
     I2C->DR = (u8)(Addr>>8);
     while((I2C->SR1&0x80)==0); //等待数据寄存器空
     I2C->DR = (u8)Addr;
     while((I2C->SR1&0x80)==0); //等待数据寄存器空
     I2C->DR = Byte_Write;
     while((I2C->SR1&0x80)==0); //等待数据寄存器空
     I2C->CR2 |= 0x02;  //产生停止信号
     i=0xffff;
     while(i--); //延时一下再读数据
//--------读AT24C64的数据-------------------------------
//1.产生I2C始起信号
     I2C->CR2 |= 0x01;  //产生始起信号
     while((I2C->SR1&0x01)==0); //等待始起信号已发送

//2.发送AT24C64的器件地址 0xA0+写操作
     I2C->DR = AT24C64_Chip;
     while((I2C->SR1&0x02)==0); //等待接收到从AT24C64返回的ACK
     Temp = I2C->SR3; //读SR3寄存器，以清除标志位ADDR

//3.发送DDR的高8位地址
     I2C->DR = (u8)(Addr>>8);
     while((I2C->SR1&0x80)==0); //等待数据寄存器空

//4.发送DDR的低8位地址
     I2C->DR = (u8)Addr;
     while((I2C->SR1&0x80)==0); //等待数据寄存器空

//5.再发送I2C始起信号
     I2C->CR2 |= 0x01;  //产生始起信号
     while((I2C->SR1&0x01)==0); //等待始起信号已发送
//6.发送AT24C64的器件地址 0xA0+读操作
     I2C->DR = AT24C64_Chip+0x01;
     while((I2C->SR1&0x02)==0); //等待接收到从AT24C64返回的ACK
     Temp = I2C->SR3; //读SR3寄存器，以清除标志位ADDR
//7.AT24C64返回数据
     while((I2C->SR1&0x40)==0); //等待数据寄存器不为空
     Byte_Read=0x00;
     Byte_Read=I2C->DR ;
     if(Byte_Read==Byte_Write){ //判断写入和读出的数据是否一至
          GPIOC->ODR &= ~(u8)0x02; //两只LED 灯亮起，表示程序运行通过
          GPIOC->ODR &= ~(u8)0x04;
     }else{
          GPIOC->ODR |= (u8)0x02; //读出和写入的数据不一致，只有LED1亮
          GPIOC->ODR &= ~(u8)0x04;
     }
     while (1); //程序停止
}
```

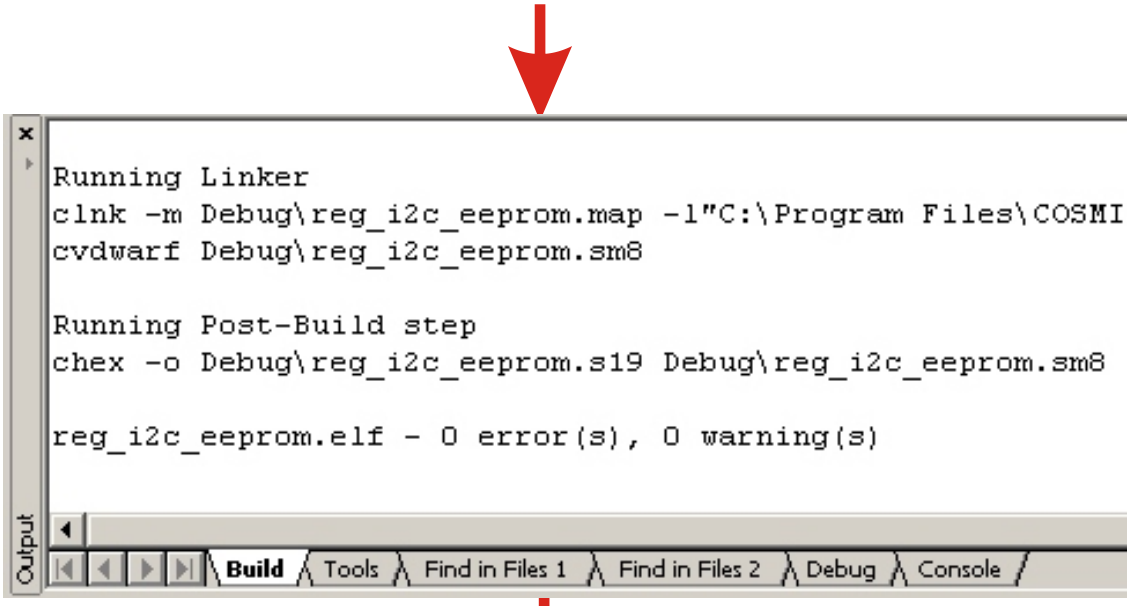
编译程序:

在编写完程序后，就可以执行编译操作了。



1. 选中工程

2. 选择build



得到. s19目标文件

下载和调试:

1.使用内置的Bootloader:

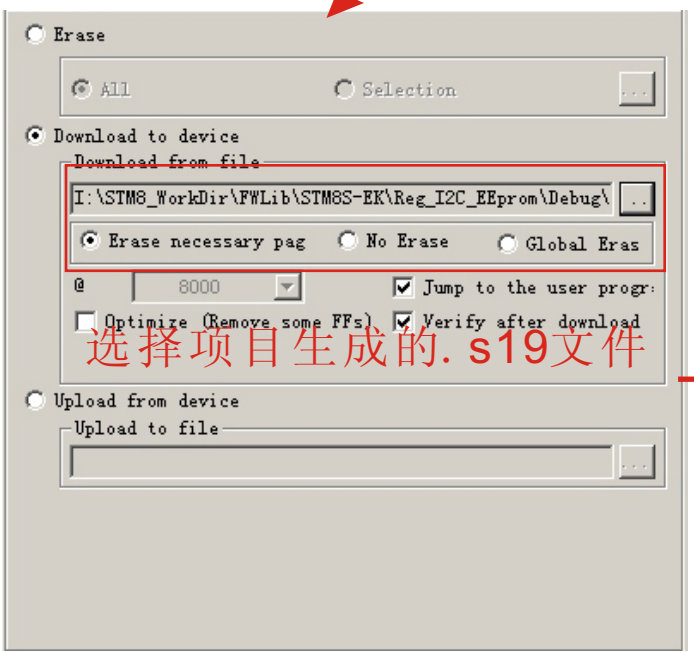
如果你没有STlink可以通过开发板上的USB接口下载程序。将编译生成的.s19目标文件下载到STM8S单片机中运行。

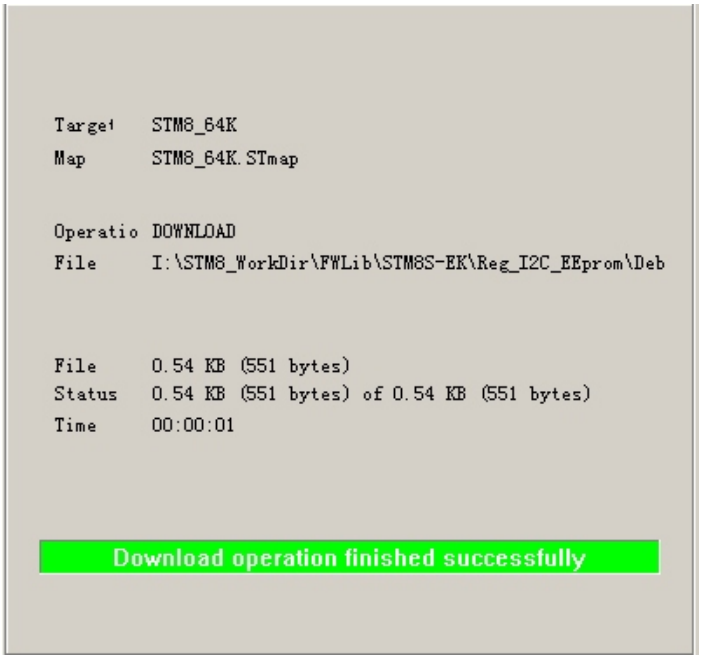
具体方法是:

1.将USB线连接PC和开发板，第一次连接要安装CP2102的驱动

2.打开STMicroelectronics flash loader 选择CP2102对应的COM口

3.按下STM8S-EK开发板的复位按键，并在1S内按下上位机的"Next"





下载完成，看看程序的运行效果吧！

STM8S-EK

作者： lisn3188
E-mail:lisn3188@163.com

STM8S-EK 开发板唯一销售网址：
http://chiplab7.taobao.com/