PIC 单片机之 I2C（从模式）实例讲解

网上有许多讲解单片机实现 I2C 主模式，但是从模式的很少。我现在就来讲讲 PIC 单片机使用 MSSP 模块实现 I2C 从模式。

有关 I2C 协议的具体介绍可以看 《PIC 单片机之 I2C(主模式)》，我们这里直接讲解实例

实例讲解：我们模仿 AT24C02 EEPROM 的协议。让一个主模式的单片机，来读取从模式单片机的数据。

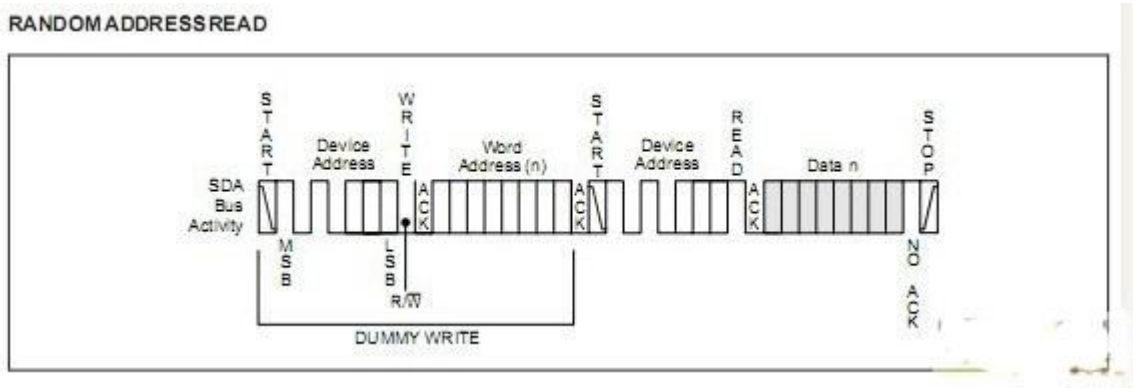
下面为 AT24C02 的随机地址读取的协议。

第一个字节 ：输入 7 位地址和一位的写状态位，

第二个字节：然后写入 EEPROM 数据地址，

第三个字节：输入 7 位地址和一位的读状态位，

第四~N 个字节：读出的 EEPROM 的数据。



我们来讲解下程序的基本思路:我们使能了 MSSP 中断,即是 I2C 接收中断，当 PIC 单片机接收到一个数据后就会产生中断。那是接收到设备地址，还是接收到数据，由 SSP1STAT 寄存器的状态位来判断。

需要判断的状态位分别是 ：

数据和地址： 用来判断接收到是地址还是数据

启动位： 用来判断是否接收到启动位

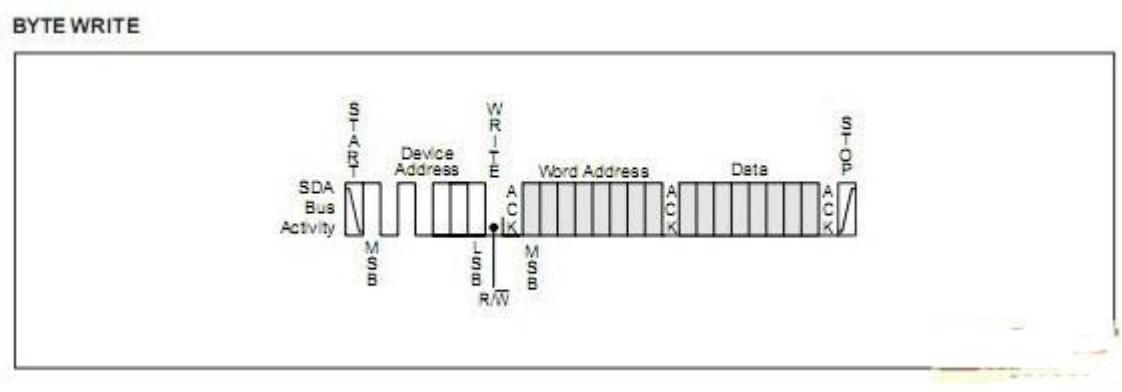
读写： 用来判断是写状态还是读状态。

缓存满： 用来判断缓冲区是否满

我们以随机地址读取为例：讲讲程序执行的过程

1，从单片机接收到启示位和设备地址中断：我们判断 SSP1STAT 的状态位为(写状态，地址，缓存满，接收到启示位) 然后读取缓存中的设备地址， 接着在读取 需要读/写的数据地址。

2，单片机再次接收到设备地址：我们判断是 SSP1STAT 的状态为(读状态) 然后从设备就输出数据



我们以写字节数据为例：

1，从单片机接收到启示位和设备地址中断：我们判断 SSP1STAT 的状态位为(写状态，地址，缓存满，接收到启示位) 然后读取缓存中的设备地址， 接着在读取 需要读/写的数据地址。

2，单片机判断 SSP1STAT 的状态位为(写状态，数据，缓存满)那么单片机就接收输入的数据。

初始化设置：

1，设置 I2C 通信的两引脚为 CLK SCL 为输入，

```
TRISB6 = input;

TRISB4 = input;
```

2，将 MSSP 设置为 I2C 从模式，七位从地址

```
SSP1CONbits.SSPM0 = 0;

SSP1CONbits.SSPM1 = 1;

SSP1CONbits.SSPM2 = 1;

SSP1CONbits.SSPM3 = 0;// I2C slave mode ,7bit address
```

3，使能 CLK 时钟

```
SSP1CONbits.CKP = 1; // enable clock
```

4,设置从设备地址为 0xA0

```
SSP1ADD =0xA0; //slave address is 0xa0
```

5，开启 I2C

```
SSP1CONbits.SSPEN=1;//enable I2c
```

6，清楚状态标志

```
SSPSTAT=0;
```

7,使能 I2C 中断

```
PIE1bits.SSP1IE = 1;//Enabe interrupt MSSP

INTCONbits.PEIE = 1;

INTCONbits.GIE = 1;
```

如果你要使用 PIC 单片机 I2C 从模式只要使用下面的代码：

将 void i2c_salve_interrupt_tx();void i2c_salve_interrupt_rx();放到中断程序中，如下：

```
void interrupt isr(void)

{

if(SSP1IE && SSP1IF)

{

i2c_salve_interrupt_tx();

i2c_salve_interrupt_rx();

SSP1IF=0;

}

}
```

将初始化函数 init_i2c_slave();放到主函数中

```
void main()
{
init_i2c_slave();
}
```

头文件 :i2c_salve.h

```
#ifndef _I2C_SALVE_H
#define _I2C_SALVE_H
void init_i2c_slave();
void i2c_salve_interrupt_tx();
void i2c_salve_interrupt_rx();
#endif
```

代码：i2c_salve.c

```
#include ;
#define input 1
#define RX_BUF_LEN 29
#define while_delay 6000
unsigned char i2c_address,word_address,Register[29];
unsigned char RANDOM_READ,i2c_counter;
extern unsigned char A_readflag;
/*I2C SALVE */
void init_i2c_slave()
{
TRISB6 = input;
```

```
TRISB4 = input;
SSP1CONbits.SSPM0 = 0;
SSP1CONbits.SSPM1 = 1;
SSP1CONbits.SSPM2 = 1;
SSP1CONbits.SSPM3 = 0;// I2C slave mode ,7bit address
SSP1CONbits.CKP = 1; // enable clock
SSP1ADD =0xA0; //slave address is 0xa0
SSP1CONbits.SSPEN=1;//enable I2c
SSPSTAT=0;
PIE1bits.SSP1IE = 1;//Enabe interrupt MSSP
INTCONbits.PEIE = 1;
INTCONbits.GIE = 1;
}
/*I2C salve mode interrupt */
void i2c_salve_interrupt_tx()//master read
{
unsigned char Temp;
unsigned int timercounter;
Temp=SSP1STAT;
Temp &= 0x2D;
if(SSP1STATbits.R_nW ==1)//Read operation.
{
A_readflag=0;
SSP1IF = 0;
```

```
i2c_address = SSP1BUF;

i2c_counter = word_address;

while(i2c_counter < RX_BUF_LEN)

{

SSP1BUF=Register[i2c_counter];//send data

SSP1CONbits.CKP=1;// enable colck

timercounter=while_delay;

while(PIR1bits.SSP1IF == 0)

{

timercounter--;

if(timercounter==0)

{

return;

}

}//waiting for ~ACK

SSP1IF = 0;

if(SSP1CON2bits.ACKSTAT == 1)

{

return ; //NOACK

}

else

{

i2c_counter++;//ACK

}
```

```
    }

    SSP1IF = 0;

    }

    }

    void i2c_salve_interrupt_rx()//master writer

    {

    unsigned char rx_status;

    unsigned char Temp;

    unsigned int timercounter;

    rx_status=false;

    Temp=SSP1STAT;

    Temp &= 0x2D;

    if(Temp==0x09)//Write operation,last byte was an address,buffer is
full

    {

    SSP1IF = 0;

    i2c_address = SSP1BUF;

    timercounter=while_delay;

    while(PIR1bits.SSP1IF == 0)

    {

    timercounter--;

    if(timercounter==0)

    {

    return ;
```

```
}
}//waiting for send ~ACK
SSP1IF = 0;
word_address = SSP1BUF;
return ;
}
if(Temp==0x29)//Write operation,last byte was data,buffer is full
{
SSP1IF=0;
Register[word_address]=SSP1BUF;
word_address++;
if(word_address>=RX_BUF_LEN)
{
word_address=0;
}
}
}
```