

```
#include <pic.h>

/*****
*   Essai de l'I2C en mode maitre esclave (2° programme Esclave)
*
* Programme équivalent à un capteur de température I2C DS1621
* a utiliser avec le programme I2C_Mast
* Fichier: (I2C_Slav2_3.c)          Version: 2.3  Du 29/03/2003
* écrit par Mr COTTET JJ Lycée Maurice GENEVOIX INGRE
* (Validée: version 1.0 à 1.2 fonctionnent)
*
* Modif pour gestion complete par interruption (V 2.0 validée)
* V2.1 Validée avec correction des LEDS
* V2.2 et 2.3 Suppression des lignes necessaires au debug (LEDs etc)
*
* Les bits RB7, RB6 et RB3 sont laissés libres pour l'interface ICD
*
* PORTC          RC4  RC3
*                SDA  SCL (du bus I2C)
*
* Fquartz=4Mhz PIC16F877 (essai avec 2 quartz à 8 MHZ sur maître et esclave)
* Développement avec MPLAB et le compilateur HI-TECH

* SSPCON = WCOL SSPOV SSPEN CKP SSPM3 SSPM2 SSPM1 SSPM0
*           0    0    1    1    0    1    1    0    pour mode I2c Slave 7 bits
* WCOL est mis à 1 lors d'une écriture dans SSPBUF si transmission en cours
*           doit être remis à 0 par logiciel
* SSPOV passe à 1 si une donnée est recue alors que SSPBUFF n'a pas été lu
*           doit être remis à 0
* SSPSTAT = SMP  CKE D/A P S R/W UA BF ; SMP=X, CKE=X, D/A=data/adresse en mode I2C
*           P=1 si stop détecté, S=1 si start détecté, R/W indique lecture ou
*           écriture (fugitif juste après la reconnaissance d'une adresse valide
*           et avant la condition de stop ou start suivante)
*           UA (seulement en adressage 10 bits)
*           BF passe à 1 si SSPBUFF est plein et doit être lu (en réception)
*           en transmission BF reste à 1 tant que SSPBUFF n'est pas fini de transmettre
* les bits sont nommés: STAT_SMP, STAT_CKE, STAT_DA, STAT_P, STAT_S, STAT_RW, STAT_UA,
*                       STAT_BF
*****/
typedef unsigned char byte; // byte en remplacement de unsigned char

/***** Définition de constantes pour l'I2C *****/
#define Ad_DS1621 0x90 // adresse par default du DS1621 (si bit ad = A2 A1 A0 = 0 0 0)
#define Capteur0 0x90
#define Capteur1 0x92
#define Access_Config 0xAC // Pour Acces au registre config du DS1621
#define Start_Convert 0xEE // Démarrage conversion du DS1621
#define Read_Temperature 0xAA // Lecture de la température (2 Octets TH et TL)
#define Conversion_continue 0x08 // Valeur du registre Config du DS1621 (Conv continue)

#define Fosc 4000000 // Fréquence d'oscillation du Quartz PIC :
#define I2CClock 100000 // Horloge I2C : 100 kHz (impulsion de l'horloge : 10us)
#define ClockValue (((Fosc/I2CClock)/4)-1)

/***** Déclaration des constantes et variables *****/

byte TempH;
byte TempL;
byte Adresse, Data, controle;
byte RW ;
byte Data_Adresse;

/***** Declaration fonctions & procedures *****/
void Init(void);
void InitI2C(void);
byte Attente_Adressage();
byte i2c_read_slave();
void i2c_write_slave(byte);

/***** routine d'interruption de l'I2C *****/
// interruptions de l'ensemble des périphérique à l'adresse 0x04
void interrupt I2C_isr(void) @ 0x04
{
```

```
    if (SSPIF)
    {
        if (STAT_DA) Data = SSPBUF;    // lecture du caractère reçu (pour remise à 0
de BF)
        else Adresse = SSPBUF;
        Data_Adresse =STAT_DA;        // INDICATEUR Data ou adresse
        RW = STAT_RW;                 // indicateur Read ou write
        SSPIF = 0;                     // acquittement événement I2C (=> ACK Slave)

        if (RW == 0)                  // ecriture du maitre (Init DS1621)
        {
            controle=i2c_read_slave();
            if (controle == Access_Config)
            {
                controle=i2c_read_slave(); // Conversion_continue
            }
            // Start_Convert: traitement inutile

            if (controle == Read_Temperature)
            {
                RW=Attente_Adressage();
                if (RW==1)
                {
                    i2c_write_slave(TempH);
                    i2c_write_slave(TempL);
                }
            }
        }

        } // fin if (RW == 0)
        SSPIF = 0;                     // acquittement événement I2C (=> ACK Slave)
    } // fin if SSPIF
}

//***** PROGRAMME PRINCIPAL *****
void main (void)
{
    Init();        // Initialiser les ports l'I2C
    InitI2C();     // Mode esclave 7 bits
    SSPIE=1; PEIE=1; GIE=1;    // autorisation des interruptions

    while(1)
    {

    }
}

//***** Fonctions (Sous programmes) *****
//***** Initialisation Ports *****
void Init(void)
{
    PORTC=0xFF;
    TRISC = 0b11111111;
    ADCON1 = 0x06; // PORTA en Digital

    TempH=35;     // pour vérifier la transmission en simulation
    TempL=0x80;   // Température simulée = 35.5 °C
}

// -----
void InitI2C(void)
{
    TRISC = 0b11111111; // RC en entrée (il faut que RC3 et RC4 soient configurés en entr
ée)
    SSPSTAT = 0b10000000; // disable slew rate, select I2C input levels
                        // STAT_CKE=0 use I2C levels, STAT_SMP=1 disable slew rate c
ontrol
    SSPADD = Capteur0; // esclave équivalent à un DS1621 ad= 0x90

    SSPCON = 0b00110110; // enable SSP, enable clock(I2C), I2C Slave Mode
    SSPCON2 = 0x00;

    SSPIF=0; // clear SSPIF interrupt flag
    BCLIF=0; // clear bus collision flag
            // SSPIE, GIE et PEIE pour autoriser les interruptions
}
```

```
        // BCLIE si on veut une interruption en cas de collision  
        // (Voir les registres INTCON, PIR1,PIE1, PIR2, PIE2)
```

```
    }
```

```
// _____ Gestion I2C _____
```

```
// *****  
***
```

```
byte Attente_Adressage()
```

```
{  
    byte temp;  
    while(SSPIF==0);           // attente activité sur le bus  
    temp = SSPBUF;             // lecture du caractère reçu (pour remise à 0 de BF)  
    if(STAT_RW==0) SSPIF = 0;  // acquittement événement I2C si adressage en écritur  
e  
    return (byte)STAT_RW;  
}
```

```
// *****  
***
```

```
byte i2c_read_slave()
```

```
{  
    byte temp;  
    while(SSPIF==0);           // attente activité sur le bus (réception d'un octet)  
    temp = SSPBUF;             // lecture du caractère reçu (pour remise à 0 de BF)  
    SSPIF = 0;                 // acquittement événement I2C  
    return temp;  
}
```

```
// *****  
***
```

```
void i2c_write_slave(byte i2c_Data)
```

```
{  
    while(SSPIF==0);           // vérification prêt à l'envoi  
    SSPBUF = i2c_Data;         // chargement de l'octet à transmettre (BF passe à 1)  
    CKP = 1;                   // deverrouillage de l'horloge SCL  
    SSPIF = 0;                 // acquittement événement I2C  
    while(SSPIF==0);           // attente fin d'envoi  
}
```