DSPIC30F3011

Bus I2C : couche liaison

Principe retenu : utilisation intensive des interruptions

1. Mode "Master"

Un seul vecteur d'interruption est associé à tous les évènements du bus. L'indicateur d'interruption MI2CIF est positionné à la fin de chaque séquence du bus : "start", "stop", transmission d'un octet, réception d'un octet, transmission de l'acquitement, réception de l'acquitement.
Le programme d'interruption est chargé de la transmission complète d'une trame préparée à l'avance. Il doit donc identifier chacune des séquences pour activer la suivante.

1.1 Sources

1.1.1 Constantes et variables

```
                        /*****************************
                        *    Définitions de type     *
                        *****************************/
typedef enum
{
  Start    = 0,
  MSend    = 1,
  MReceive = 2,
  Restart  = 3,
  MAck     = 4,
  Stop     = 5
}tSequenceI2C; // Type de séquence I2C

typedef enum
{
  Write    = 0,
  Read     = 1
}tR_W; // Type "R/W" pour accès I2C

                        /*****************************
                        * Constantes non mémorisées  *
                        *****************************/
#define FCY 4000000*16/4  // xtal = 4Mhz; PLLx16 -> 16 MIPS

                        /*****************************
                        *       Variables            *
                        *****************************/
/* Liaison I2C
**************/
tSequenceI2C SequenceI2C_act; // Code de la séquence I2C actuelle
tR_W R_W_I2C; // Type d'accès I2C : Write ou Read
unsigned int Nb_octets_I2C; // Nombre d'octets à transmettre à partir de
                            // l'adresse pDatas_I2C
unsigned int Cpt_octets_I2C;// Compteur d'octets transmis sur le bus I2C
unsigned char *pDatas_I2C;  // Utilisé comme pointeur vers les données I2C
```

1.1.2 Initialisation du coupleur I2C

```
/*****************************************************************
  Function:    void InitI2C(void)
  Description: Initialisation du coupleur I2C :
                - mode Master
                - horloge SCL = 50kHz
                - interruptions "Master" validées
*****************************************************************/
#define FSCL 50000 // Pour 50kHz
void InitI2C(void)
{
 I2CBRG=FCY/FSCL-FCY/1111111-1; // FSCL=50kHz
 I2CCON=0x8000; // I2CEN=0 : broches 26=SDA, broche 25=SCL
                // I2CSIDL=0 : module actif en mode "Idle"
                // SCLREL=0 : sans effet
                // IPMIEN=0 : mode IPMI inhibé
                // A10M=0 : adresse I2C sur 7 bits
                // DISSLW=0 : slew rate control validé
                // SMEN=0 : seuils d'entrée SMBus inhibés
                // GCEN=0 : pas d'appel général
                // STREN=0 : "clock stretching" inhibé
                // ACKDT=0 : transmission ACK durant acquitement
                // ACKEN=RCEN=PEN=RSEN=SEN=0 : aucune séquence I2C activée
}
```

1.1.3 Fonction d'écriture

```
/*****************************************************************
  Function:    void Write_to_I2C(void)
  Description: écriture de "Nb_octets_I2C" pointés par "pDatas_I2C" dans une
               EEPROM I2C
  Pré-requis : Les octets pointés par "pDatas_I2C" doivent être affectés.
               En particulier, le 1° est l'adresse I2C de la cible (déjà
               décalée d'un cran à gauche).
               "Nb_octets_I2C" est affectée
  Traitement : - activation de la séquence START
               - le programme d'interruption _MI2CInterrupt s'occupe des autres
                 séquences pour écrire les octets ds l'E2P
*****************************************************************/
void Write_to_I2C(void)
{
 R_W_I2C=Write;    // Ecriture ds l'E2P
 Cpt_octets_I2C=0; // Raz compteur d'octets (utilisé par _MI2CInterrupt)
 SequenceI2C_act=Start; // Séquence I2C actuelle
 I2CCONbits.SEN=1; // Activation de la séquence START
                   // Une interruption est activée par MI2CIF à la fin
                   // Le programme d'interruption _MI2CInterrupt se charge du
                   // reste du traitement
}
```

1.1.4 Fonction de lecture

```
/*****************************************************************
  Function:    void Read_I2C_to_RXD(void)
  Description: lecture de "Nb_octets_I2C" depuis l'EEPROM I2C et transfert
               vers TXD
  Pré-requis : Les premiers octets pointés par "pDatas_I2C" doivent être
               affectés par l'adresse I2C de la cible (déjà décalée d'un
               cran à gauche) et l'adresse de base à lire ds l'EEPROM.
               La variable "Nb_octets_I2C" est affectée
  Traitement : - activation de la séquence START
               - le programme d'interruption _MI2CInterrupt s'occupe des autres
                 séquences pour lire les Nb_octets_I2C ds l'E2P et les copier
                 dans le buffer TXD
*****************************************************************/
void Read_I2C_to_RXD(void)
{
 R_W_I2C=Read;          // Lecture de l'E2P I2C
 SequenceI2C_act=Start; // Séquence I2C actuelle
 Cpt_octets_I2C=0; // Raz compteur d'octets (utilisé par _MI2CInterrupt)
 I2CCONbits.SEN=1; // Activation de la séquence START
                   // Une interruption est activée par MI2CIF à la fin
                   // Le programme d'interruption _MI2CInterrupt se charge du
                   // reste du traitement
}
```

1.1.5 Fonction d'interruption

```
/*****************************************************************
  Function:        void _ISR _MI2CInterrupt (void)
  Description :    Activé à chaque fin de séquence I2C
                   Traitements :
                    - En écriture :
                      - 1ière activation à la fin de la séquence START
                      - puis transmission des "Nb_octets_I2C" à partir de
                        pDatas_I2C sur le bus I2C
                      - activation séquence STOP à la fin ou si NACK
                    - En lecture :
                      - 1ière activation à la fin de la séquence START, puis
                      - transmission adresse I2C en écriture,
                      - transmission adresse E2P à lire,
                      - activation RESTART,
                      - transmission adresse I2C en lecture,
                      - puis lecture de "Nb_octets_I2C" sur le bus I2C
                        et transfert sur TXD
                      - activation séquence STOP à la fin ou si NACK
*****************************************************************/
void _ISR _MI2CInterrupt (void)
{
 IFS0bits.MI2CIF=0; // Acquitement interruption
 if (R_W_I2C==Write)
 {
```

```
  // Accès en écriture
  // *****************
  switch (SequenceI2C_act)
  {
   case Start : // Fin de la séquence START ?
   {
    SequenceI2C_act=MSend;// Oui : début de la séquence de transmission d'octets
    I2CTRN=*pDatas_I2C   ;// Début séquence de transmission de l'adresse I2C en WR
                          // Prochaine interruption à la réception de l'acquitement
    pDatas_I2C++; // Pour pointer l'adresse E2P
    break;
   }
   case MSend : // Réception d'un acquitement de transmission d'octet ?
   {
    Cpt_octets_I2C++; // Incrémentation du compteur d'octets transmis
    pDatas_I2C++;
    if (I2CSTATbits.ACKSTAT)
    {
     SequenceI2C_act=Stop; // NACK : arrêt du transfert
     I2CCONbits.PEN=1;     // Activer la séquence STOP
     break;
    }
    if (Cpt_octets_I2C < Nb_octets_I2C) // Octet de données ?
    {
     I2CTRN=*pDatas_I2C;  // Oui : recherche de l'octet suivant
     break;
    }
    if (Cpt_octets_I2C==Nb_octets_I2C) // Dernier octet transmis ?
    {
     SequenceI2C_act=Stop; // Arrêt du transfert
     I2CCONbits.PEN=1;     // Activer la séquence STOP
     break;
    }
   }
   case Stop : // Fin de la séquence STOP ?
   {
    break; // Oui : on ne fait rien !
   }
  }
 }
 else
 {
  // Accès en lecture
  // ****************
  switch (SequenceI2C_act)
  {
   case Start : // Fin de la séquence START ?
   {
    I2CTRN=*pDatas_I2C; // Début séquence de transmission de l'adresse I2C en WR
                        // Prochaine interruption à la réception de l'acquitement
    pDatas_I2C++;       // Pour pointer l'adresse E2P
    SequenceI2C_act=MSend;// Début de la séquence de transm. de l'adresse
    break;
   }
   case MSend : // Fin de la transmission d'un octet ?
   {
    if (I2CSTATbits.ACKSTAT) // Transmission acquitée ?
    {
     SequenceI2C_act=Stop; // NACK : arrêt du transfert
     I2CCONbits.PEN=1;     // Activer la séquence STOP
     break;
    }
    Cpt_octets_I2C++; // Incrémentation du compteur d'octets transmis
    pDatas_I2C++;     // Pour pointer l'octet LSB de l'adresse E2P
    if (Cpt_octets_I2C < 3) // Transmission adresse E2P ?
```

```
    {
     I2CTRN=*pDatas_I2C;  // Oui : recherche de l'octet suivant
     break;
    }
    if (Cpt_octets_I2C==3)  // Adresse complète transmise ?
    {
     SequenceI2C_act=Restart; // Oui : séquence RESTART
     I2CCONbits.RSEN=1;       // Activer la séquence RESTART
     break;
    }
    if (Cpt_octets_I2C==4)  // Fin transmission adresse I2C après Restart ?
    {
     SequenceI2C_act=MReceive;// Oui :  début séquence "réception"
     Cpt_octets_I2C=0;  // Pour compter les octets lus
     I2CCONbits.RCEN=1; // Activer la séquence de réception d'un octet
     break;
    }
   }
   case Restart : // Fin séquence Restart ?
   {
    I2CCONbits.ACKDT=0; // Acquitement pour tous les octets lus, sauf le dernier
    I2CTRN=DATASCONT.Adr_I2C+1;// Après Restart : transmission adresse I2C en RD
    SequenceI2C_act=MSend;     // Prochaine interruption : test ACK
    break;
   }
   case MReceive : // Fin de réception d'un octet ?
   {
    Cpt_octets_I2C++;    // Incrémentation du compteur d'octets lus
    WriteTXD(I2CRCV);    // Octet reçu transféré dans buffer TXD
    if (Cpt_octets_I2C==Nb_octets_I2C) I2CCONbits.ACKDT=1; // NACK pour le dernier
octet
    I2CCONbits.ACKEN=1;  // Activer la séquence ACK
    SequenceI2C_act=MAck;// Prochaine interruption à la fin de la séquence
    break;
   }
   case MAck : // Fin d'une séquence ACK ?
   {
    if (Cpt_octets_I2C==Nb_octets_I2C)
     {
      SequenceI2C_act=Stop; // Arrêt du transfert à la lecture du dernier octet
      I2CCONbits.PEN=1;     // Activer la séquence STOP
     }
    else
     {
      SequenceI2C_act=MReceive;// Nouvelle séquence "réception"
      I2CCONbits.RCEN=1; // Activer la séquence de réception d'un octet
     }
    break;
   }
   case Stop : // Fin de la séquence STOP ?
   {
    break; // Oui : on ne fait rien !
   }
  }
 }
}
```

Algorithmes

_MI2CInterrupt

Acquitement de l'interruption MI2CIF

Ecriture (J) ou lecture (N) en cours ?

J	N
Ecriture d'un octet vers un "esclave"	Lecture d'un octet depuis un esclave

Ecriture d'un octet vers un esclave

Ecriture d'un octet vers un esclave I2C

Start	SequenceI2C_act ? (quel type de séquence est terminée ?) MSend			
SequenceI2C_act <- MSend	Incrémenter le compteur d'octets Cpt_Octets_I2C			Stop
I2CTRN <- octet pointé par pDatas_I2C	Incrémenter le pointeur pDatas_I2C pour pointer l'octet suivant			
Incrémenter le pointeur pDatas_I2C pour pointer l'adresse I2C	Acquitement de la cible ? J		N	
	Dernier octet transmis ? J		N	
	SequenceI2C_act <- Stop		SequenceI2C_act <- Stop	
	Déclenchement de la séquence Stop		Déclenchement de la séquence Stop	
		I2CTRN <- octet pointé par pDatas_I2C (début de la transmission de l'octet suivant)		

Lecture d'un octet depuis un esclave