Le BUS I2C

Le protocole I2C

APPLICATIONS

C. Majastre Décembre 2004

Sommaire

APPLICATIONS

1) EXEMPLE de LECTURE I2C.

Le DALLAS 1621 :

L'utilisation du thermomètre digital Dallas 1621 donne un exemple de lecture de plusieurs octets en I2C.

La trame envoyée doit commencer par l'adressage du boîtier avec un accès en écriture, suivi du code de début de conversion.

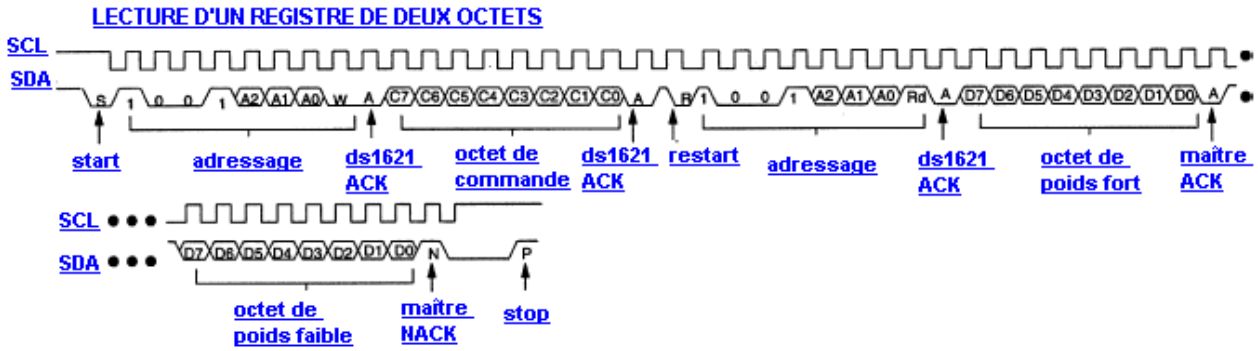
Il faut laisser un bref délai pour que le thermomètre effectue une première conversion.

Ensuite il faut commencer une nouvelle communication en s'adressant au boîtier toujours en écriture. L'octet suivant contient le code de la fonction de lecture.

Une fois ces opérations effectuées on crée un "restart" (condition de stop immédiatement suivie d'une condition de start), on s'adresse à l'esclave en mode lecture et on attend la réponse.

ds1621 ACK: acquittement de l'esclave

Cette réponse est sur deux octets. On effectuera donc un acquittement après réception du premier octet et après réception du deuxième octet un non-acquittement suivi d'une condition de stop.



2) EXEMPLE d'ECRITURE I2C :

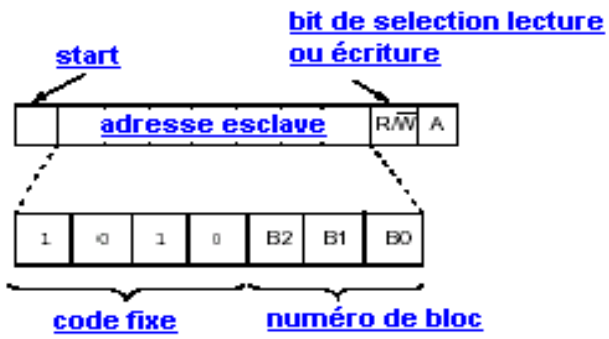
2-1) L'EEPROM 24LC16 :

L'écriture dans une mémoire EEPROM en mode I2C suit le même principe que la lecture des registres du Dallas1621.
Il faut suivre le protocole I2C (start, stop...) et envoyer les informations suivantes:

-L'adresse du boîtier
-L'adresse de l'octet
-Les données à écrire

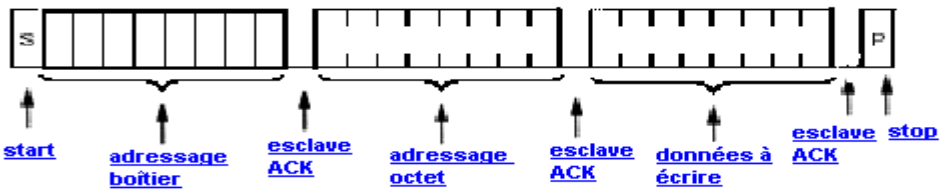
L'EEPROM 24lc16 à la particularité d'avoir l'adresse boîtier codée uniquement sur les quatre bits de poids fort. Le dernier bit codant le mode d'accès (lecture/écriture). Il reste trois bits disponibles qui servent à coder l'adresse d'un des huit blocs internes.

Octet d'adressage

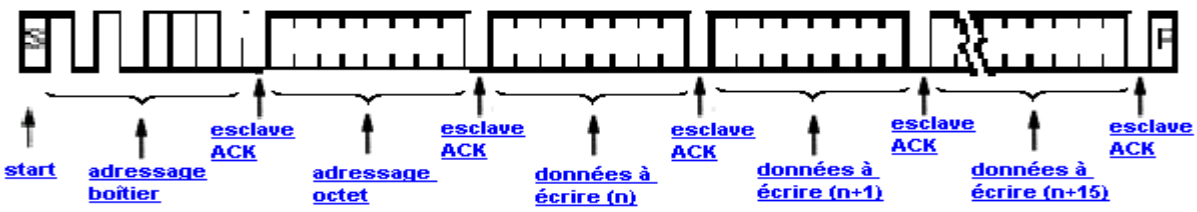


Ces huit blocs contiennent chacun 256 octets mémoires dont l'adresse est envoyée dans le second octet transmis.

Ecriture d' un octet

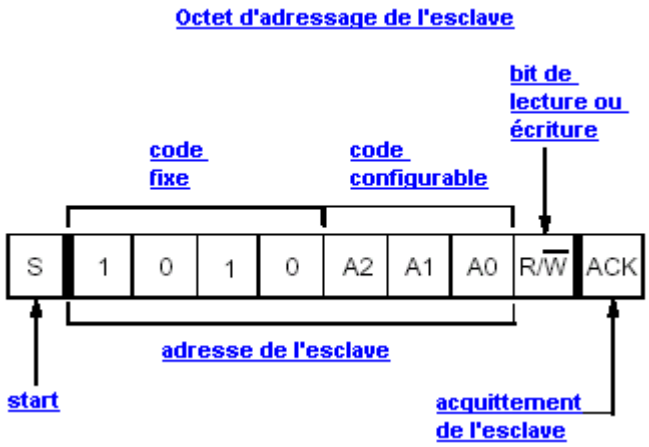


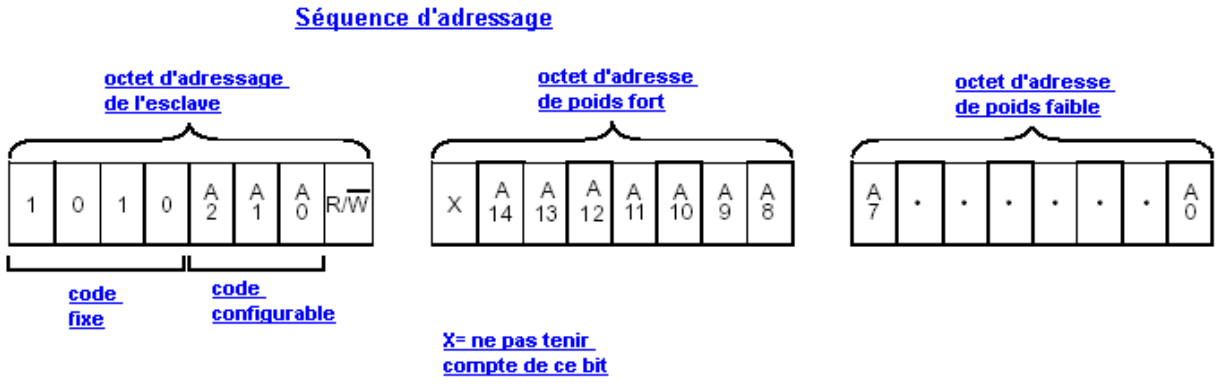
Ecriture de n octets



2-2) L'EEPROM 24lc256 :

Elle fonctionne selon le même principe que L'EEPROM 24lc16 à la différence que l'adresse boîtier est codée sur 7 bits dont trois sont configurables par l'utilisateur. L'adresse interne est codée sur deux octets envoyés immédiatement après l'adresse du boîtier.





Excepté la séquence d'adressage, L'EEPROM 24lc256 utilise le même format de trame que la mémoire 24lc16.

3) L'I2C ET LE PIC 16F877 :

Le PIC16F87x possède des entrées/sorties spécialement prévues pour la communication I2C.

Ce sont les broches nommées **RC3 (SCL) et RC4 (SDA).**

Elles sont directement reliées à un contrôleur I2C.

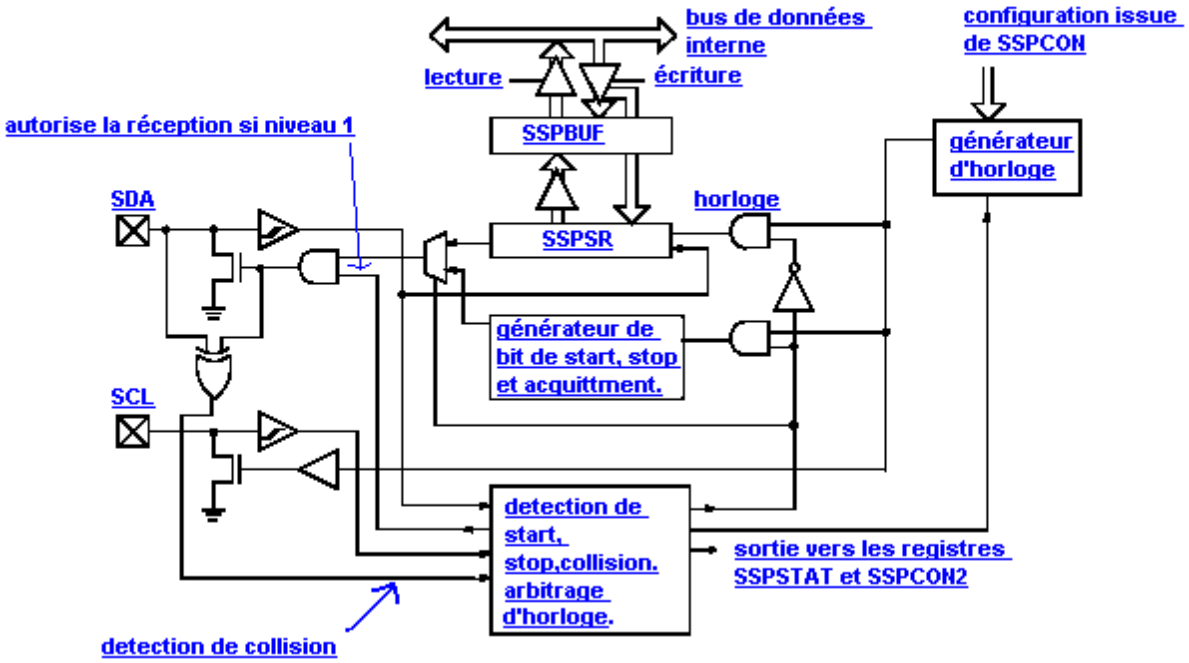
Ce contrôleur prévoit la gestion de tout le protocole I2C (Prise de parole, collision, ligne occupée...). Il permet une gestion facile de la liaison.

Avant d'utiliser le contrôleur il faut paramétrer quatre registres :

SSPSTAT, SSPCON, SSPCON2, SSPADD.

Ces registres servent à définir la vitesse de communication, l'horloge source...

SCHEMA DU CONTROLEUR I2C



Une fois les registres de configuration mis à jour, Quelques commandes simples suffisent pour formater une trame :

-La commande « **SEN** » crée une condition de start.

-La commande « **RSEN** » crée une condition de restart.

-La commande « **PEN** » crée une condition de stop.

-Les octets transmis ou reçus sont stockés dans le buffer « **SSPBUF** »

-Les fins d'opération sont testées avec la variable « **SSPIF** »

-L'acquittement ou non-acquittement sont paramétrables avec les variables « **ACKDT** » **et** « **ACKEN** ».

Pour communiquer au protocole I2C, il suffit de formater la trame au modèle fourni par le constructeur du composant choisi et utiliser ces quelques commandes.

Exemple de programme PIC/I2C :

```
void init (void)
{       TRISA4  = 0;
        ADCON1  =0x06;
        TRISC3  = 0;        // sortie horloge SCL.
        TRISC4  = 1;        // Ligne SDA.
        TRISB0  = 1;
        TRISB1  = 0;
        SSPSTAT = 0x80;  // Pas de contrôle de slew rate, niveau I2C.
        SSPCON  = 0x38;  // Validation I2C sur 7 bits Master Mode.
        SSPCON2 = 0x00;
        SSPADD  = 9;        // Fclock=1OOKHz Fosc/4(SSPAD+1).
        rejet   = SSPBUF;   // Vide le buffer et RAZ des Flags.
        BRGH=1 ;
        SPBRG   = 25;
        GIE     = 0;
        PEIE    = 0;
        TXSTA   = 0x24;
        TXREG   = 0x00;
        RCSTA   = 0x90;
}

void LC_write_un (char bt,char oth,char otl,char mt)
{       SEN   = 1;
        while (SEN == 1);
        SSPIF = 0;
        SSPBUF= (bt & 0xFE);
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= oth;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= otl;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= mt;
        while (SSPIF == 0);
        SSPIF = 0;
        PEN   =1;
        while(PEN);
}
```

4) EXPERIMENTATIONS :

Nous avons utilisé trois circuits différents en communication I2C :

-Le convertisseur de température Dallas 1621
-La mémoire EEPROM 24LC16
-La mémoire EEPROM 24LC256

On utilise un analyseur logique pour l'observation des états physiques des composants mis en jeu. Il s'utilise sur PC avec le logiciel LA VIEWER.
Il permet d'effectuer un relevé des communications entre le PIC16F876 et les différents composants I2C.
Cette possibilité d'observer physiquement les communications apporte une aide précieuse dans la résolution des problèmes.
Cela permet de trouver rapidement les origines des dysfonctionnements :

- Mauvais format de trame.
- Délais d'attente trop bref entre deux communications.
- Composant défectueux...

4-1) Le convertisseur de température :

4-1-1) Fonctionnement :

Le capteur de température Dallas 1621 possède une adresse définie sur 7 bits. De ces sept bits, les quatre de poids forts sont déterminés par le constructeur alors que les trois de poids faibles sont au choix de l'utilisateur.

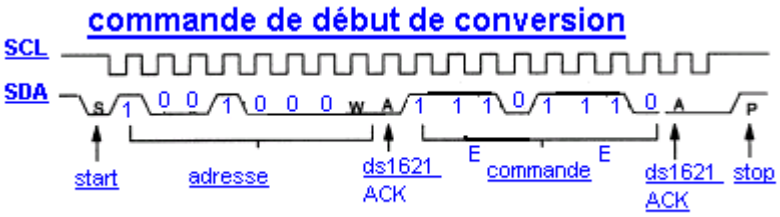
Ce composant contient un capteur de température analogique.
La valeur donnée par le capteur est convertie en une valeur numérique codée sur neuf bits.

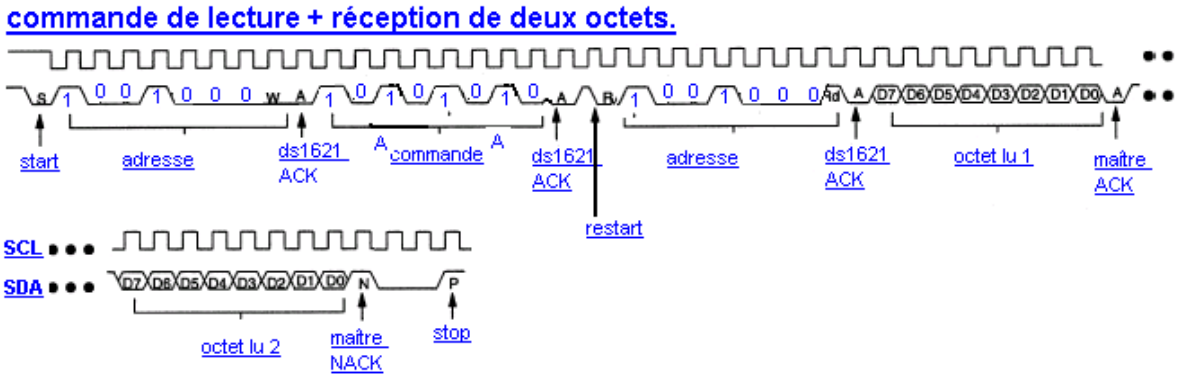
Cette conversion est effectuée en continue dès lors que le DALLAS 1621 a reçu l'ordre de commencer la conversion (code commande : 0xEE).

Relation entre la valeur numérique et la température

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	01111101 00000000	7B00h
+25°C	00011001 00000000	1900h
+½°C	00000000 10000000	0080h
+0°C	00000000 00000000	0000h
-½°C	11111111 10000000	FF80h
-25°C	11100111 00000000	E700h
-55°C	11001001 00000000	C900h

On peut aussi stopper cette conversion.
Une fois la conversion lancée, il ne reste plus qu'à faire des demandes de lecture pour obtenir la valeur convertie (code commande 0xAA).

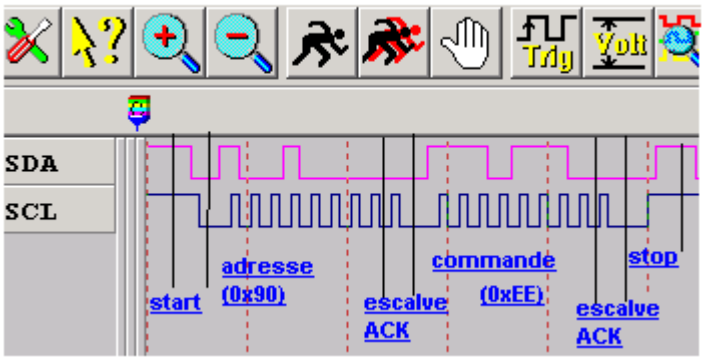




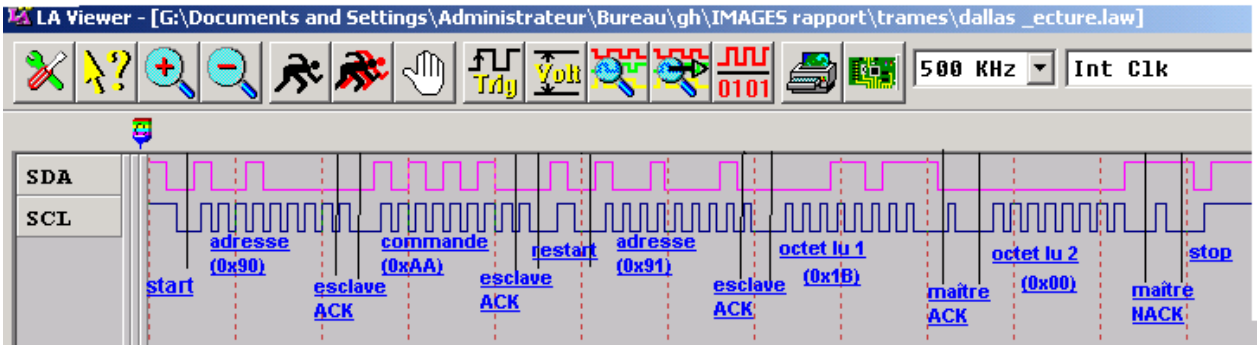
4-1-2 Expérimentation :

Une fois le capteur connecté correctement (les broches SDA et SCL du capteur reliées aux broches RC4 et RC3 du PIC...) et les registres configurés, il suffit de programmer les trames à envoyer au capteur selon le format ci-dessus. En se connectant aux lignes SDA et SCL avec l'analyseur logique, on peut observer l'état physique des lignes, et le bon déroulement de la communication.

Envoie de la commande de début de conversion



Ecriture de la commande de lecture + réception des deux octets lus



4-2) La mémoire EEPROM 24lc16 :

4-2-1) Fonctionnement :

Comme tous les composants I2C, L'EEPROM 24lc16 doit être alimentée et avoir ses broches SCL et SDA connectées.
Son principe de fonctionnement est simple :

Pour écrire des données dans la mémoire il suffit d'adresser le boîtier en mode écriture et d'envoyer les deux octets d'adresse interne.
Ensuite il ne reste plus qu'à envoyer les données à écrire.

Le composant incrémente seul l'adresse après chaque octet écrit, ce qui permet de cumuler les données à écrire sans recommencer une nouvelle communication.

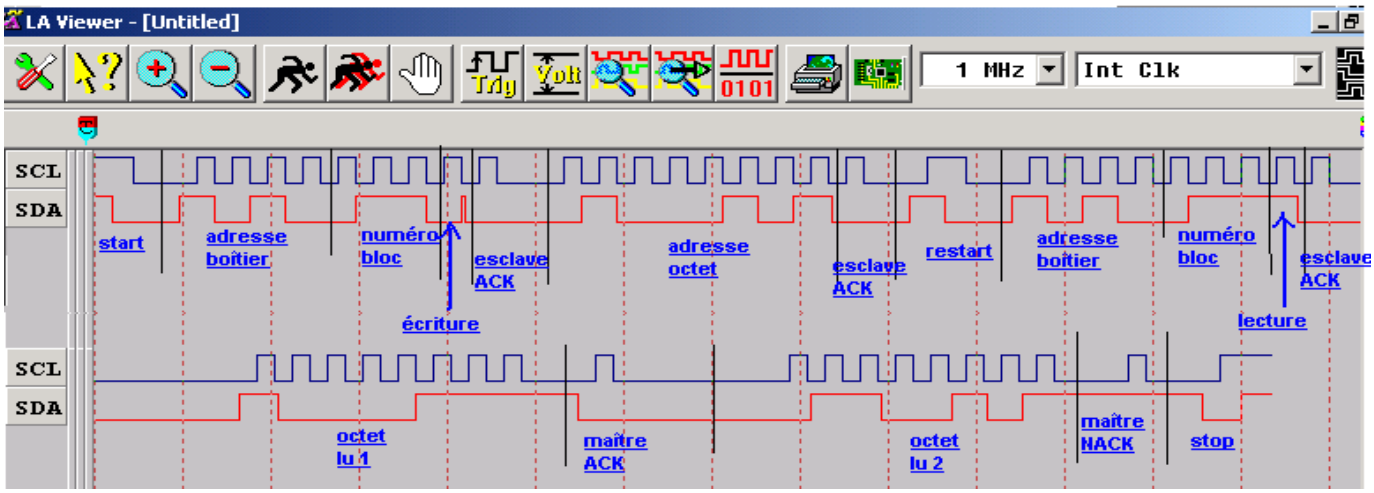
4-2-2) Expérimentation :

On envoie une trame d'écriture de plusieurs octets et on vérifie le bon fonctionnement en effectuant ensuite une demande de lecture à la même adresse.
L'espionnage de trame nous permet de contrôler le déroulement de l'opération.
Puis on fait une demande d'écriture dans le boîtier répondant à l'adresse 0xA6.

Cette demande est affectée à l'octet d'adresse 0x45 du bloc numéro 2 (trame ci-dessus).

Afin de vérifier que l'action a bien été effectuée, on fait une demande de lecture à la même adresse.
La fin de lecture est déterminée par le maître qui crée un non-acquittement suivi d'une condition de stop.
La lecture a bien renvoyé les octets qui ont été écrits au préalable.
(trame ci-dessous).

Lecture de deux octets dans une mémoire 24LC16



4-3) La mémoire EEPROM 24lc256 :

4-3-1) Fonctionnement :

La mémoire 24lc256 répond aux mêmes commandes que la 24lc16 exception faîte de l'adressage qui diffère :

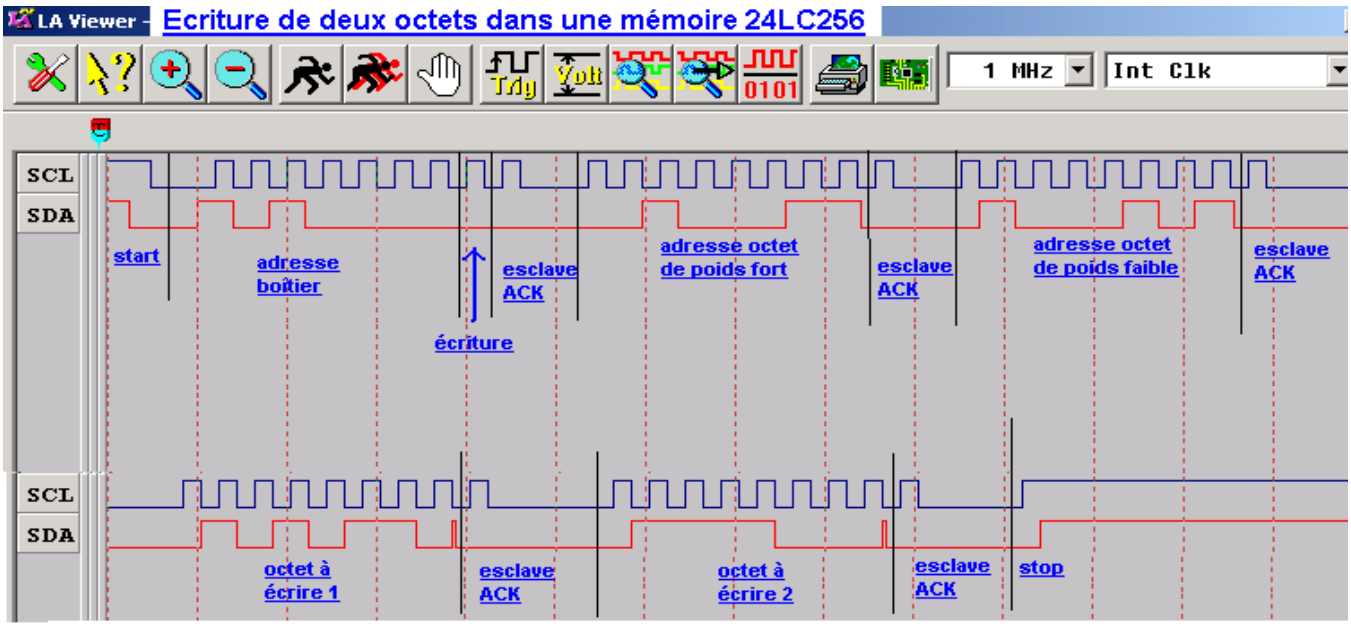
- L'adresse du boîtier est cette fois codée sur 7 bits.
- Il n'y a plus d'affectation de bloc.
- L'adresse octet est codée sur 14 bits (donc envoyée sur deux octets).

4-3-2) Manipulation :

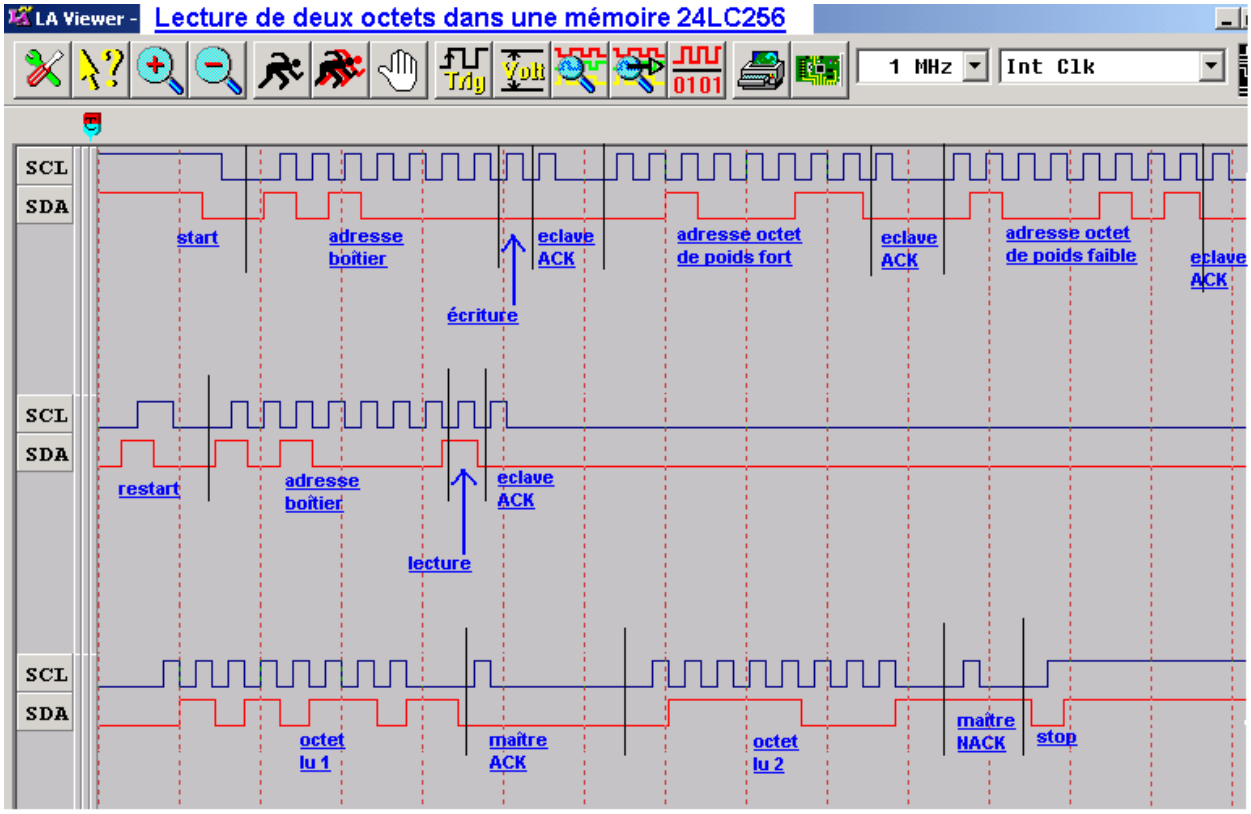
En réalisant les mêmes opérations que précédemment :

- Demande d'écriture de deux octets à une adresse quelconque.
- Demande de lecture de deux octets à la même adresse.

En demandant l'écriture des octets 0x56 et 0x78 dans les cases mémoires à partir de l'adresse : 0x2345 et ce dans le boîtier répondant à l'adresse boîtier 0xA0.
Voici la trame obtenue à l'aide de l'analyseur logique.



En faisant ensuite fait une demande de lecture de deux octets à cette même adresse, et nous obtenons la trame ci-dessous :



5) CONCLUSION :

L'I2C est un protocole de communication rapide et facile à mettre en oeuvre.
Il permet de créer des bus contenant plusieurs maîtres et esclaves.
Ce protocole prévoit la gestion automatique des conflits entre composants.
(Ce sont les composant eux-même qui s'en occupent).
Cette étude se limite au fonctionnement en mode mono-maître.
En revanche le protocole ne prévoit aucune vérification des données transmises.
Il est alors difficile de savoir si les valeurs reçues sont exactes.

6) PROGRAMME I2C COMPLET:

```
/* Definition des bibliotheques*/

#include <pic.h>
#include "biosdem.h"

/* configuration */

#define MAX 0xF100
  __CONFIG(0x3539);

/* Declaration des fonctions */

void LC_read (char,char,char,char);
void LC_write_un (char,char,char,char);
void LC_write_plus (char,char,char,char);
void dallas (void);
void init (void);
void emis(char);
char recep(void);

/* Declaration des variables */

bank1 char RW[]="lecture=1 ; ecriture=0";
bank1 char ADB[]="adresse boitier";
bank2 char ADOL[]="adresse octet low";
bank2 char NBO[]="nombre d'octet";
bank2 char MAE[]="mot(s) pour ecriture";
bank2 char ADOH[]="adresse octet high";
char i,i2,i3,rejet,t;
bank3 char
mr,mr2,nbo,adoh1,adoh2,adol1,adol2,bt,ot,ado2,rw,adb,adb2,ml[10],ml2[10],tem
p,temp2,temp3,temp4,adbl,mae[9];

/* Programme principal */

void main (void)
{
        init();
        for( ; ; )
```

```
{
        for(i=0;i<22;i++)
        {
                emis(RW[i]);
        }
        emis(0x0A);
        emis(0x0D);
        rw=recep();                             /* demande RW*/
        for(i=0;i<15;i++)
        {
                emis(ADB[i]);
        }
        emis(0x0A);
        emis(0x0D);
        adb=recep();                            /* demande boîtier*/
        adb2=recep();
        adb=(adb << 4);
        adb=(adb | adb2);
        if(adb==0x90)
        {
                dallas();
                temp2=(temp & 0x0F);
                temp=(temp>>4);
                temp4=(temp3 & 0x0F);
                temp3=(temp3>>4);
                if((temp>=0x00)&&(temp<0x0A))
         {
                        temp= temp+0x30;
                }
                if((temp>0x09)&&(temp<0x10))
                {
                        temp=temp+0x37;
                }
                if((temp2>=0x00)&&(temp2<0x0A))
         {
                        temp2= temp2+0x30;
                }
                if((temp2>0x09)&&(temp2<0x10))
                {
                        temp2=temp2+0x37;
                }
                if((temp3>=0x00)&&(temp3<0x0A))
         {
```

```
                temp3= temp3+0x30;
            }
            if((temp3>0x09)&&(temp3<0x10))
            {
                temp3=temp3+0x37;
            }
            if((temp4>=0x00)&&(temp4<0x0A))
          {
                temp4= temp4+0x30;
            }
            if((temp4>0x09)&&(temp4<0x10))
            {
                temp4=temp4+0x37;
            }
            emis(temp);
            emis(temp2);
            emis(temp3);
            emis(temp4);
            emis(0x0A);
            emis(0x0D);
        }
        if(adb==0xA0)
        {
            for(i=0;i<18;i++)
            {
                emis(ADOH[i]);
            }
            emis(0x0A);
            emis(0x0D);
            adoh1=recep();      /* demande adresse octet high si lc256 */
            adoh2=recep();
            adoh1=((adoh1 << 4) | adoh2 );
            for(i=0;i<17;i++)
            {
                emis(ADOL[i]);
            }
            emis(0x0A);
            emis(0x0D);
            adol1=recep();                  /* demande adresse octet low
*/
            adol2=recep();
            adol1=((adol1<<4) | adol2);
            for(i=0;i<14;i++)               /* demande nb octet*/
```

```
{
        emis(NBO[i]);
}
emis(0x0A);
emis(0x0D);
nbo=recep();                        /* demande nb mots*/
if(rw==0)
{
        for(i=0;i<21;i++)
        {
                emis(MAE[i]);
        }
emis(0x0A);
emis(0x0D);
        for(i=0;i<nbo;i++)
        {
        mr=recep();
        mr2=recep();
        mae[i]=((mr<<4)| mr2);  /* si ecriture demande mots*/
        }
        if(nbo==1)
        {
                LC_write_un(adb,adoh1,adol1,mae[0]);
        }
        if(nbo>1)
        {
                LC_write_plus(adb,adoh1,adol1,nbo);
        }
}
if(rw==1)
{
        LC_read(adb,adoh1,adol1,nbo);
        for(i=0;i<nbo;i++)
        {
                ml2[i]=(ml[i] & 0x0F);
                ml[i]=(ml[i] >> 4);
                if((ml[i]>=0x00)&&(ml[i]<0x0A))
        {
                        ml[i]= ml[i]+0x30;
                }
                if((ml[i]>0x09)&&(ml[i]<0x10))
                {
                        ml[i]=ml[i]+0x37;
```

```
                                }
                                if((ml2[i]>=0x00)&&(ml2[i]<0x0A))
                        {
                                        ml2[i]= ml2[i]+0x30;
                                }
                                if((ml2[i]>0x09)&&(ml2[i]<0x10))
                                {
                                        ml2[i]=ml2[i]+0x37;
                                }
                                emis(ml[i]);
                                emis(ml2[i]);
                        }
                        emis(0x0A);
                        emis(0x0D);
                }
        }
    }
}

/* Fonctions */

void init (void)
{
    TRISA4  = 0;
    ADCON1  =0x06;
    TRISC3  = 0;
    TRISC4  = 1;
    TRISB0  = 1;
    TRISB1  = 0;
    SSPSTAT = 0x80;
    SSPCON  = 0x38;
    SSPCON2 = 0x00;
    SSPADD  = 9;
    rejet   = SSPBUF;
    SPBRG   = 25;
  GIE     = 0;
    PEIE    = 0;
  TXSTA   = 0x24;
    TXREG   = 0x00;
    RCSTA   = 0x90;
}

void emis(char cae)
```

```
{
        while(!TXIF);
        TXREG   = cae ;
}

char recep (void)
{
        char cp;
        while (!RCIF);
        if((RCREG>0x2F)&&(RCREG<0x3A))
        {
                cp=RCREG-0x30;
        }
        if((RCREG>0x40)&&(RCREG<0x47))
        {
                cp=RCREG-0x37;
        }
        return cp;
}


void dallas (void)
{
        SEN   = 1;
        while (SEN == 1);
        SSPIF = 0;
        SSPBUF= 0x90;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= 0xEE;
        while (SSPIF == 0);
        RSEN  = 1;
        while (RSEN == 1);
        SSPIF = 0;
        SSPBUF= 0x91;
        while (SSPIF == 0);
        SSPIF = 0;
        RCEN  = 1;
        while(STAT_BF == 0);
        temp  = SSPBUF;
        ACKDT = 1;
        ACKEN = 1;
        while(ACKEN == 1);
```

```
        SSPIF = 0;
        PEN   = 1;
        while(PEN == 1);

        SEN   = 1;
        while (SEN == 1);
        SSPIF = 0;
        SSPBUF= 0x90;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= 0xAA;
        while (SSPIF == 0);
        RSEN  = 1;
        while (RSEN == 1);
        SSPIF = 0;
        SSPBUF= 0x91;
        while (SSPIF == 0);
        SSPIF = 0;
        RCEN  = 1;
        while(STAT_BF == 0);
        temp  = SSPBUF;
        ACKDT = 0;
        ACKEN = 1;
        while(ACKEN == 1);
        SSPIF = 0;
        RCEN  = 1;
        while(STAT_BF == 0);
        temp2 = SSPBUF;
        ACKDT = 1;
        ACKEN = 1;
        while(ACKEN == 1);
        SSPIF = 0;
        PEN   = 1;
        while(PEN == 1);
}

void LC_read (char b,char oh ,char ol,char n)
{
                SEN   = 1;
                while(SEN);
                SSPIF = 0;
                SSPBUF= (b & 0xFE);
                while(!SSPIF);
```

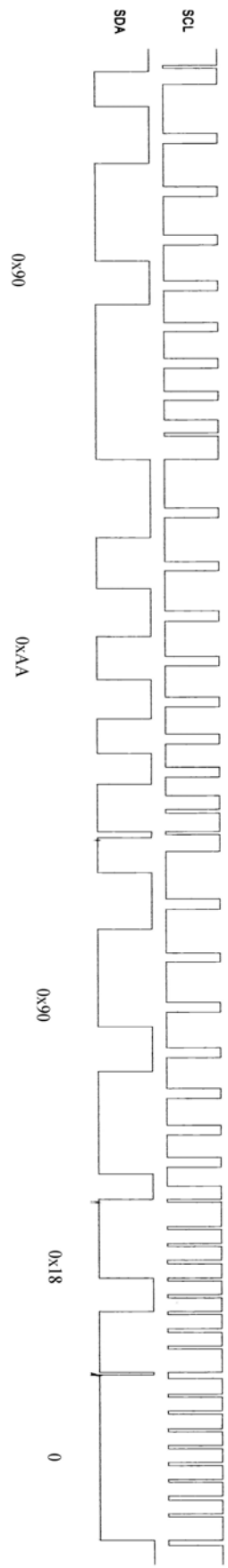
```
                SSPIF = 0;
                SSPBUF= oh;
                while(!SSPIF);
                SSPIF = 0;
                SSPBUF= ol;
                while(!SSPIF);
                SSPIF = 0;
                RSEN  = 1;
                while(RSEN);
                SSPIF = 0;
                SSPBUF= (b | 0x01);
                while(!SSPIF);
                for(t=0;t<100;t++)
                { }
        for(i2=0;i2<(n-1);i2++)
        {
                RCEN  = 1;
                while(STAT_BF == 0);
                ml[i2] = SSPBUF;
                ACKDT = 0;
                ACKEN = 1;
                while(ACKEN == 1);
                SSPIF = 0;
        }
        RCEN  = 1;
        while(STAT_BF == 0);
        ml[(n-1)] = SSPBUF;
        ACKDT = 1;
        ACKEN = 1;
        while(ACKEN == 1);
        SSPIF = 0;
        PEN   = 1;
        while(PEN == 1);
        DelayMs(250);
}

void LC_write_un (char bt,char oth,char otl,char mt)
{
        SEN   = 1;
        while (SEN == 1);
        SSPIF = 0;
        SSPBUF= (bt & 0xFE);
        while (SSPIF == 0);
```

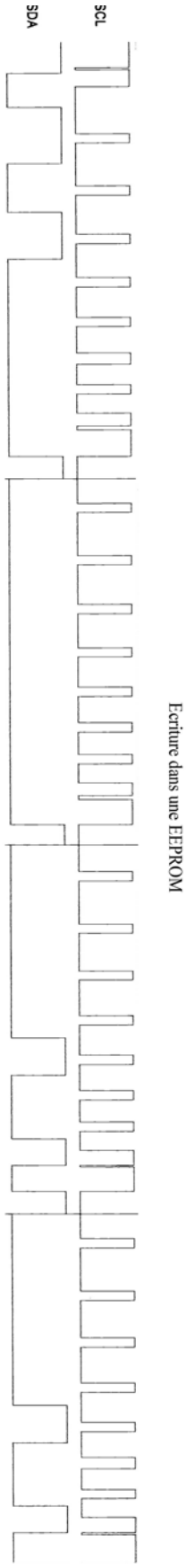
```
        SSPIF = 0;
        SSPBUF= oth;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= otl;
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= mt;
        while (SSPIF == 0);
        SSPIF = 0;
        PEN   =1;
        while(PEN);
}


void LC_write_plus (char boi,char octh,char octl,char nb)
{
        SEN   = 1;
        while (SEN);
        SSPIF = 0;
        SSPBUF= (boi & 0xFE);
        while (!SSPIF);
        SSPIF = 0;
        SSPBUF= octh;
        while (!SSPIF);
        SSPIF = 0;
        SSPBUF= octl;
        while (!SSPIF);
        for(i3=0;i3<nb;i3++)
        {
                SSPIF = 0;
                SSPBUF= mae[i3];
                while (!SSPIF);
        }
        SSPIF = 0;
        PEN   =1;
        while(PEN);
        DelayMs(5);
}
```

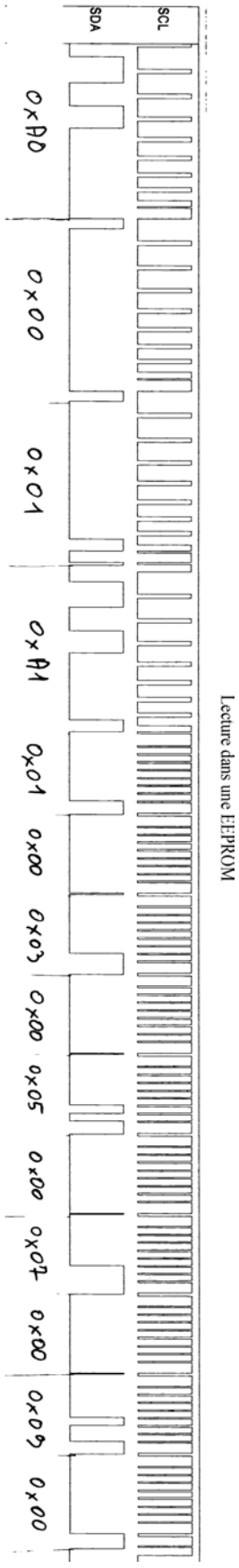
7) TRAME I2C : CAPTEUR DALLAS 1621.



8) TRAME I2C MEMOIRE EEPROM 24lc256.



Ecriture dans une EEPROM



Lecture dans une EEPROM

9) DOCUMENTATION TECHNIQUE :

Capteur de température DALLAS/ MAXIM DS 1621.

Mémoire EEPROM 16K ISSI 24LC16.

Mémoire EEPROM 64K Microchip 24LC64.