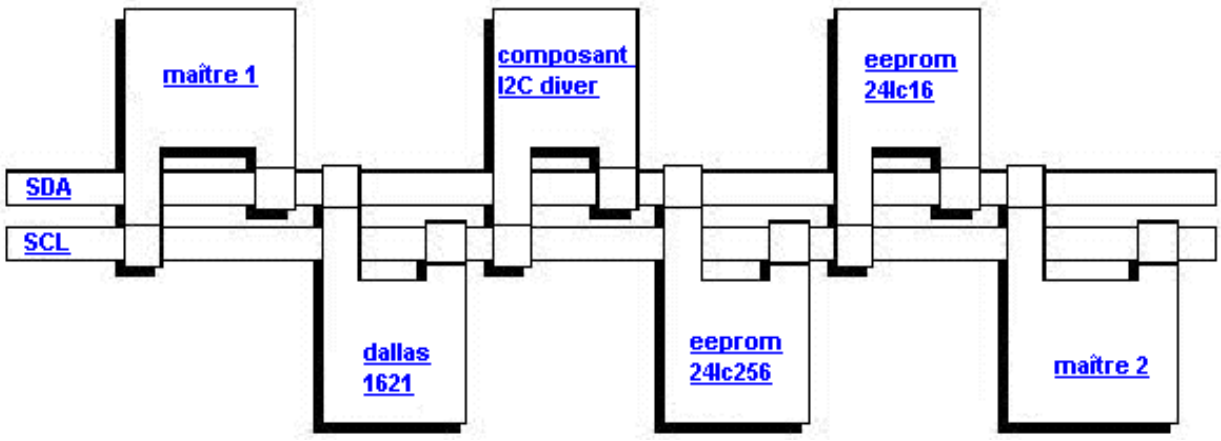
LE BUS I2C

Le protocole I2C

1- Présentation

L'I2C (Inter Integrated Circuit bus) a été créé par PHILIPS dans les années 80.
Il a été adopté par un grand nombre de constructeurs.
L'I2C est un protocole de communication de niveau physique.
Son fonctionnement s'appui sur trois fils (SDA : Serial Data, SCL : Serial Clock et la référence : Masse).
Le fil nommé SCL transmet l'horloge pour la synchronisation de communication; alors que la ligne SDA transmet les informations (adresses et données).
Un seul boîtier maître peut commander plusieurs boîtiers esclaves, (chacun d'entre eux répondant à une adresse unique).
Un esclave ne s'exprime que sur ordre d'un maître et plusieurs maîtres peuvent partager le même réseau.

Exemple de bus I2C



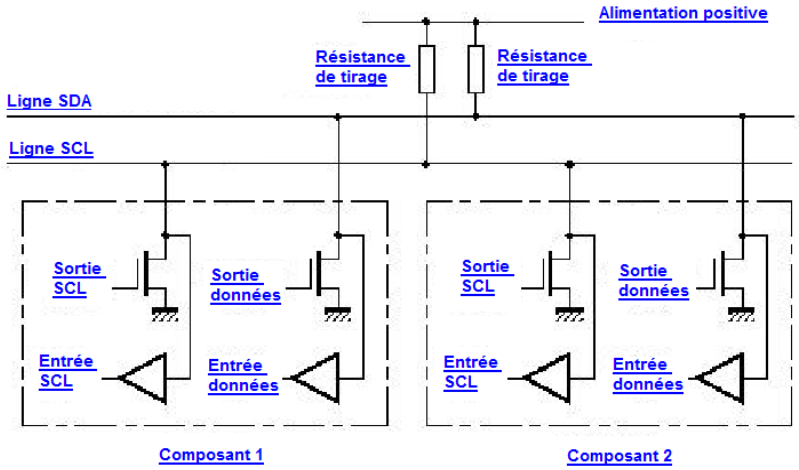
Le protocole I2C prévoit la gestion de prise de contrôle des lignes et des collisions de données. (Nous travaillerons en « mono-maître », il est donc inutile de nous occuper de ce type de problèmes).

2- Le protocole

2-1 Configuration matérielle

Les lignes SDA et SCL sont bidirectionnelles (entrée/sortie), de ce fait pour éviter tout conflit électrique elles sont de type « collecteur ou drain ouvert » et sont câblées à travers une résistance de tirage vers l'alimentation positive du circuit.

Le câblage réalise ainsi sur chaque ligne un "ET logique".



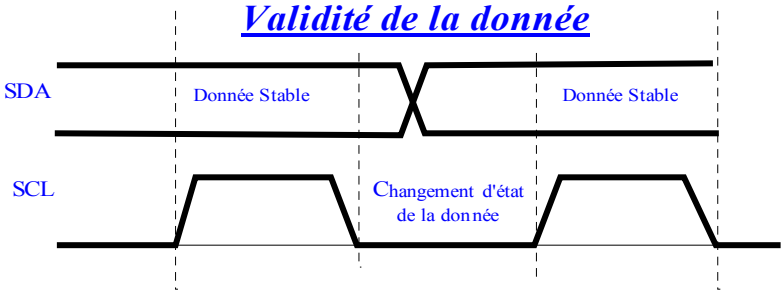
Au repos ces lignes sont à l'état « haut », le bus est dit "libre".

2-2) La transmission de données

Validité de la donnée

La transmission étant de type série, les données sont transmises bit par bit, avec pour chaque bit transmis, un coup d'horloge.

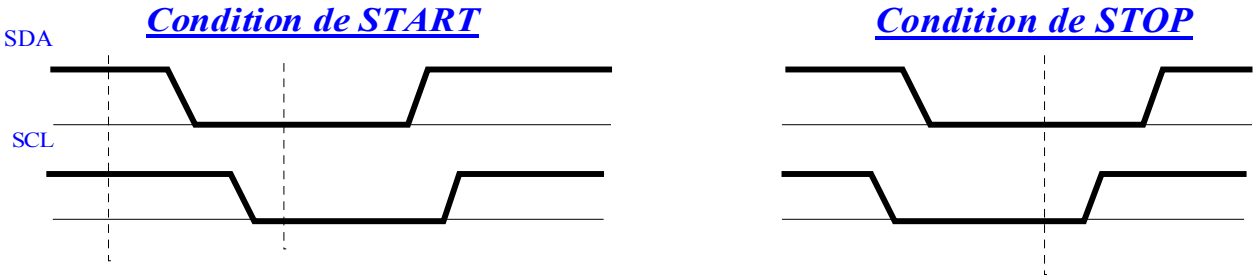
Une donnée est valide que si la ligne d'horloge SCL est à un niveau haut et que la ligne de donnée SDA est stable. Les changements d'état de la ligne SDA doivent être effectués lors d'un état bas sur la ligne SCL.



Conditions de START et de STOP

Un début de transmission doit respecter des conditions particulières :
La ligne SDA passe de l'état haut à l'état bas alors que la ligne SCL est à l'état haut.

Une fin de transmission a aussi un caractère particulier :
La ligne SDA passe de l'état bas à l'état haut alors que la ligne SCL est à l'état haut.



C'est entre ces deux conditions que les données doivent être transmises.

FORMATS DE TRANSMISSION

Format des mots

Chaque mot transmis sur la ligne SDA a une longueur de 8 bits. Le bit de poids fort (MSB) est transmis d'abord.

Formats des transferts

Principe de base

Chaque transfert commence par la **condition de départ** suivie des adresses et des données**.**
Chaque octet est transmis du bit de poids fort au bit de poids faible.

Chaque mot transmis doit être suivi par un bit d'acquittement **ACK (acknowledge),** c'est le circuit qui reçoit la donnée qui doit mettre à l'état bas la ligne SDA pour transmettre l'acquittement ACK.

Si le récepteur veut interrompre la transmission il force l'horloge au niveau bas. Lorsqu'elle sera libérée l'émetteur pourra continuer sa transmission de données.

Chaque transfert se termine par la condition d'arrêt **STOP**.

Transmission d'un octet

La transmission d'un mot (octet) se fait bit à bit en commençant par le bit de poids fort B7 que l'émetteur (le maître) applique sur la ligne SDA. Puis il valide la donnée en appliquant pendant un instant un niveau haut sur la ligne SCL. Lorsque SCL revient au niveau bas, il recommence l'opération jusqu'à ce que l'octet complet soit transmis. Il y a alors acquittement de la part du récepteur. (Voir paragraphe acquittement).

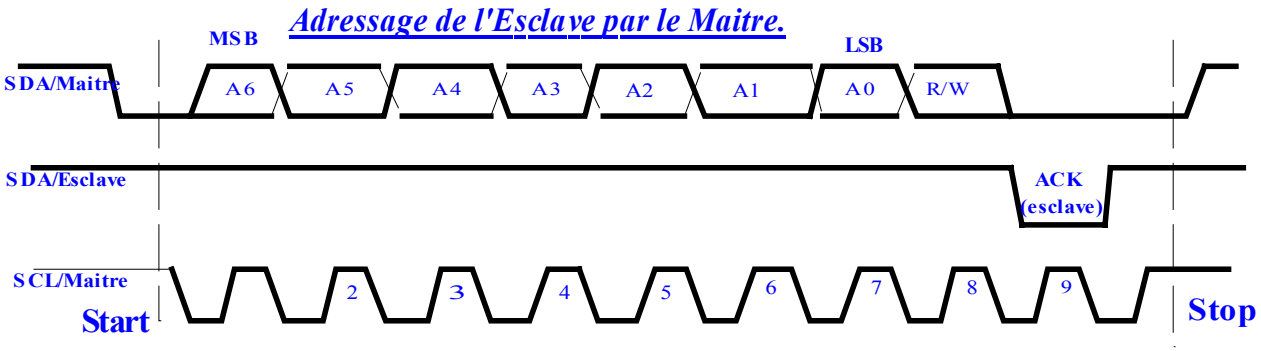
Signification des mots pendant la transmission

Premier mot transmis

Le premier mot transmis est l'adresse de l'esclave (codée sur 7 bits), le huitième bit (LSB) est le bit de lecture/ écriture : R/W.
Si ce bit est à 0 : c'est une écriture du maître vers l'esclave.
Si ce bit est à 1 : c'est une lecture de l'esclave vers le maître.

Après la condition de départ tous les récepteurs lisent le premier octet émis pour savoir si cela correspond à leur adresse, puis s'il est sélecté suivant l'état du bit R/W se considèrent comme esclave récepteur (qui va lire la donnée) ou comme esclave émetteur qui va écrire des données sur le bus.

Nota : Les informations SDA Maître et SDA Esclave sont véhiculées sur le fil unique SDA. Ce fonctionnement est rendu possible grâce au ET logique.



ACQUITTEMENT

Le transfert de données avec acquittement est obligatoire, à la fin de chaque mot transmis.

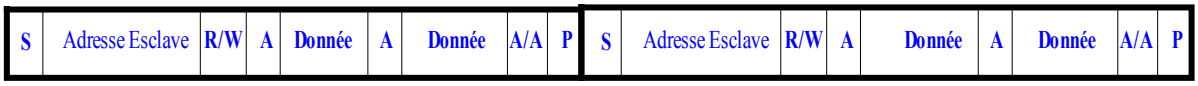
A la fin de chaque transmission d'un octet par le maître, l'esclave doit imposer un niveau bas **(ACK)** sur la ligne SDA pendant que la ligne SCL est à l'état haut. Ceci pour signaler au maître que la transmission s'est effectuée correctement. Les sorties des lignes SDA et SCL étant à collecteurs ouverts, le maître détecte le niveau bas et peut alors continuer le transfert de l'octet suivant.

Ceci implique donc que le maître doit laisser libre la ligne SDA pendant que la ligne d'horloge SCL est à l'état haut à la fin de la transmission du mot.

NON ACQUITTEMENT

Si l'esclave ne positionne pas la ligne SDA à l'état bas (ACK) : cas d'un esclave occupé à une autre tâche prioritaire, le maître peut alors générer une condition d'arrêt.

3- Exemple de transmissions sur la ligne série SDA :



S: Start.

A : Acknowledge (acquitement)

$\overline{A}$/A : Acquitement SDA=0, Non Acquitement SDA=1

P : Stop

$\overline{R}$/W: = 1 : Lecture d'une donnée de l'esclave vers le maitre ;
= 0 : Ecriture d'une donnée du maitre vers l'esclave

LECTURE / ECRITURE DE DONNEES

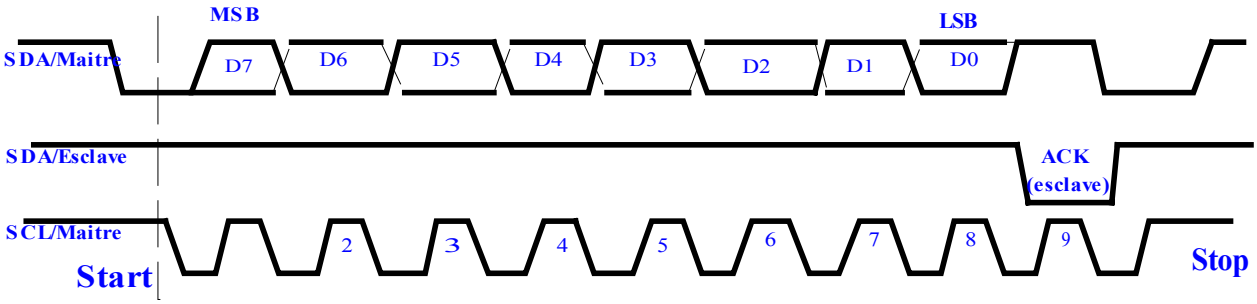
Ecriture d'une donnée

Le maître émet des données vers un esclave :

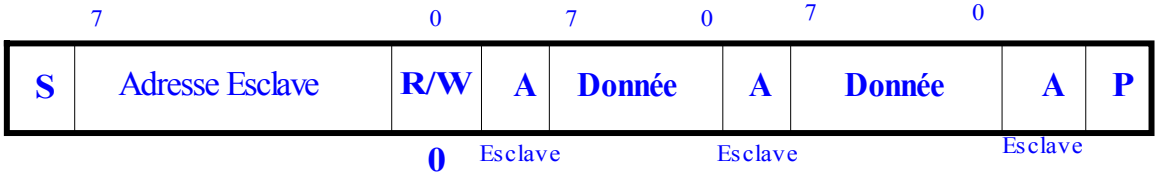
Le maître, a une donnée à transmettre vers l'esclave, l'écriture de cette donnée se fait par la ligne SDA en synchronisation avec l'horloge de la ligne SCL.

L'esclave doit acquitter la lecture de la donnée en positionnant sur la ligne SDA le signal ACK. L'écriture d'un octet dans certains composants (mémoires, microcontrôleur, ..) peut prendre un certain temps. Il est donc possible que le maître soit obligé d'attendre l'acquittement ACK.

Nota : Les informations SDA Maître et SDA Esclave sont véhiculées sur le fil unique SDA. Ce fonctionnement est rendu possible grâce au ET logique.



Transfert de données du maître vers l'esclave (Ecriture du Maître)

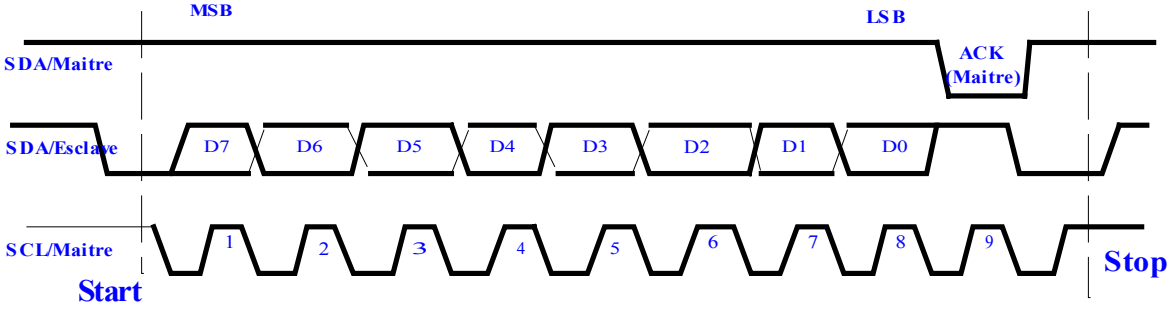


Lecture d'une donnée

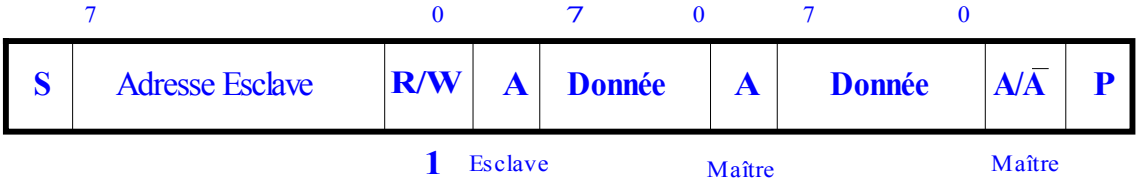
Le maître lit des données en provenance d'un esclave

Le maître demande une lecture d'une donnée vers l'esclave (cas d'un circuit mémoire dans lequel il faut lire la donnée). Ceci se caractérise par une utilisation spéciale du signal ACK. Après la lecture d'un octet, le maître positionne ACK à l'état bas. S'il veut lire la donnée suivante (dans la mémoire par exemple), ou à l'état haut pour une fin de transmission. Il envoie alors la condition d'arrêt.

Lecture d'une donnée : Esclave->Maitre



Transfert de données de l'esclave vers le maître (Lecture du maître)



2-5 Exemple de lecture

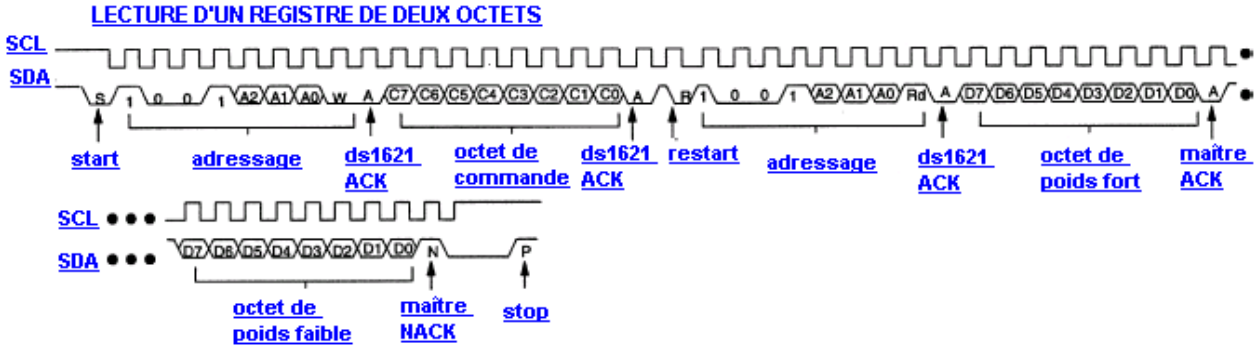
Le DALLAS 1621

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.
Cette opération de restart est obligatoire dans le cas d'un passage du mode écriture en mode lecture sans interruption de la trame.
Cette dernière 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.



Nota: L'adresse sur 7 bits est constituée par :

- 4 bits de poids forts fixés par le constructeur.
- 3 bits de poids faibles configurables par câblage (8 boîtiers différents sur le même bus).

2-6) Exemple d'écriture :

L'EEPROM 24LC16 :

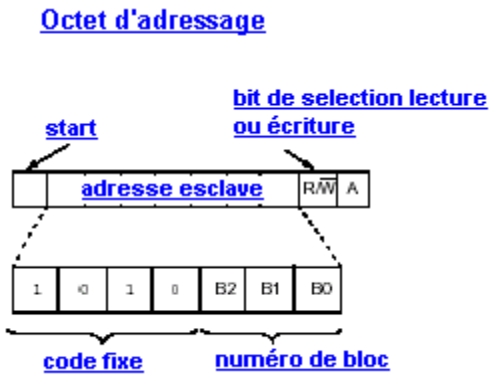
L'écriture dans une mémoire EEPROM en mode I2C suit le même principe que la lecture des registres du Dallas1621.
Il faut suivre le protocole I2C (start, stop...) et envoyer les informations suivantes:

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

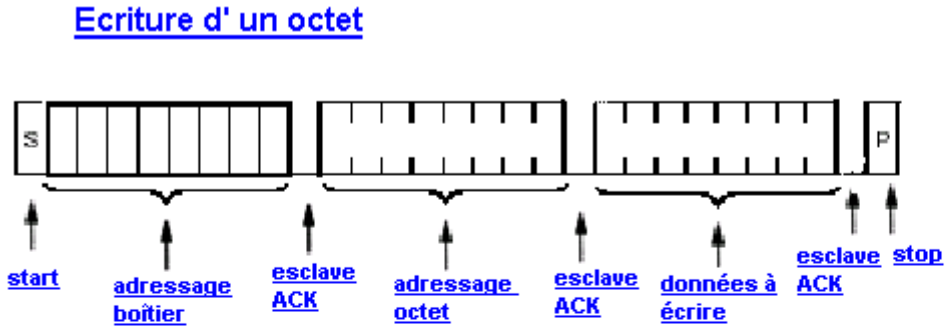
L'EEPROM 24lc16 a la particularité d'avoir l'adresse boîtier codée uniquement sur les quatre

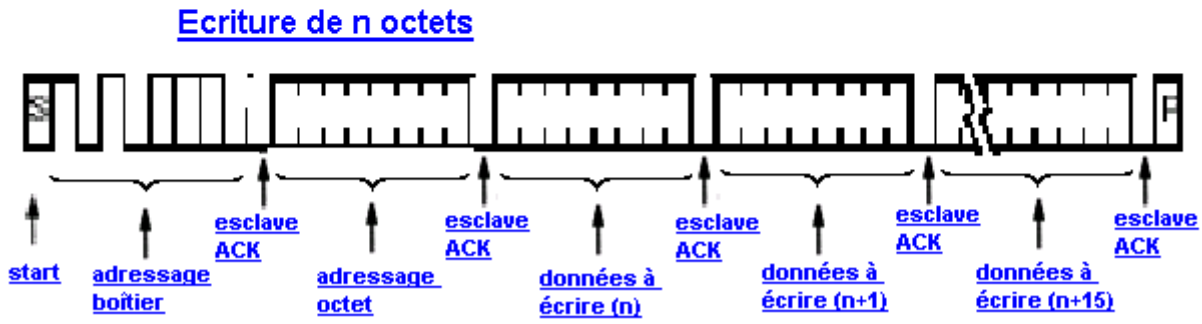
bits de poids fort. Le dernier bit codant le mode d'accès (lecture/écriture), il reste trois bits disponibles qui servent à coder l'adresse d'un des huit circuits utilisables.

Adresse boîtier :



Ces huit circuits possibles contiennent chacun 256 octets mémoire dont l'adresse est envoyée dans le second octet transmis.



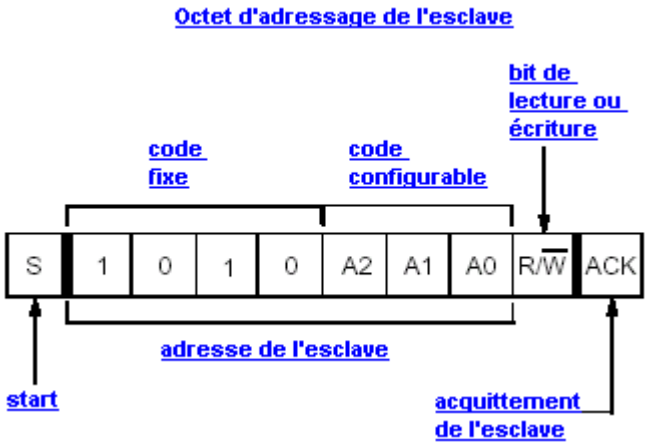


L'EEPROM 24lc256

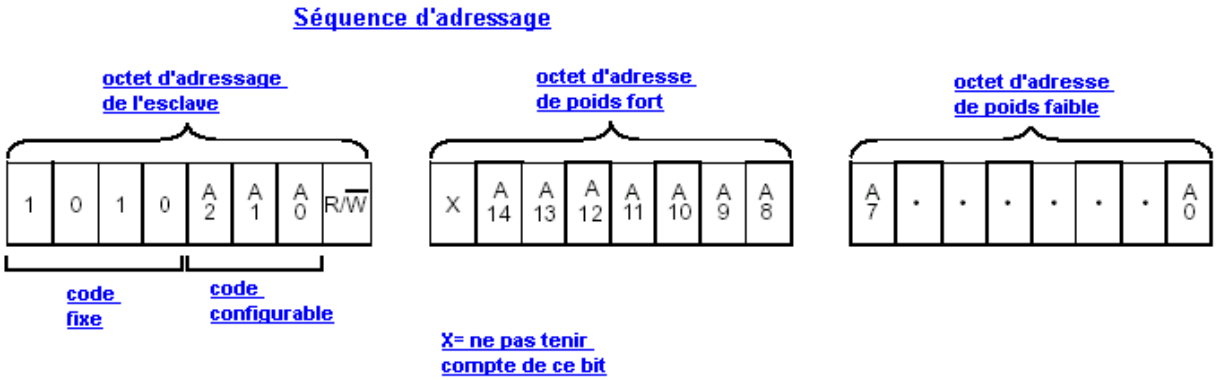
Elle fonctionne selon le même principe que L'EEPROM 24lc16 : l'adresse boîtier est codée sur 7 bits dont trois sont configurables par l'utilisateur.

Par contre sa taille étant de 4 Ko, il est nécessaire d'utiliser deux octets pour adresser un octet de donnée.

Adresse boîtier :



L'adresse interne d'une donnée est codée sur deux octets envoyés immédiatement après l'adresse du boîtier.

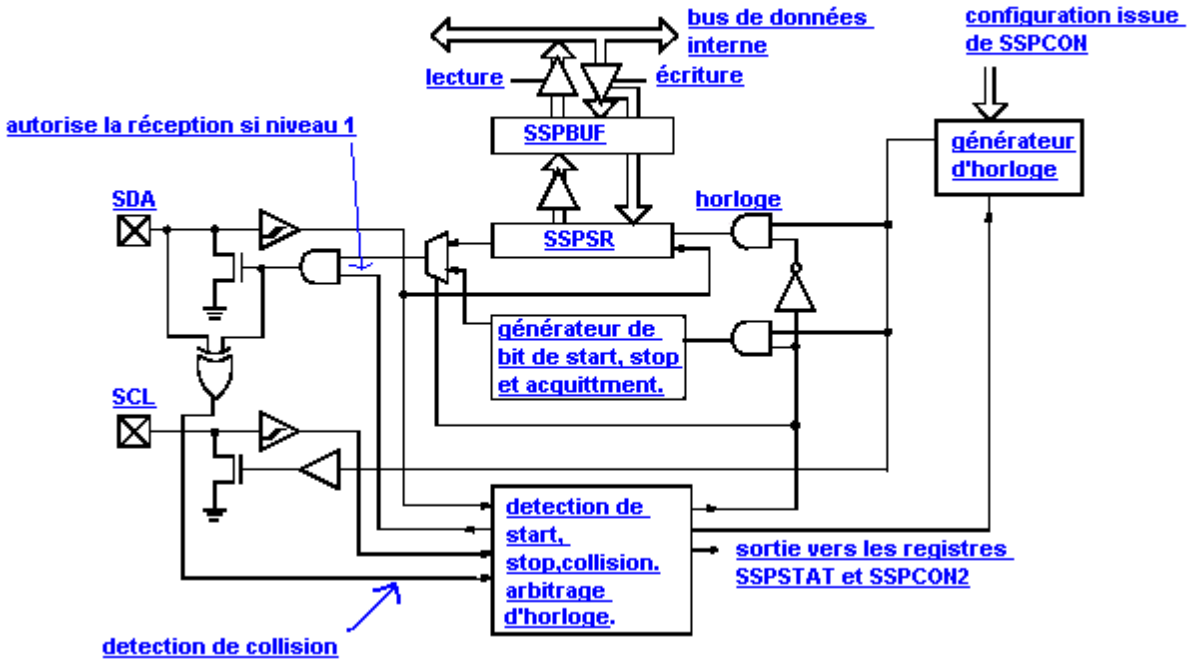


Excepté la séquence d'adressage ; L'EEPROM 24lc256 utilise le même format de trame que la mémoire 24lc16.

3- L'I2C et le PIC 16F87x :

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 à transmettre 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 en utilisant ces quelques commandes.

4) Les registres du contrôleur I2C:

Le Registre SSPSTAT (Registre d'état)

Bit SMP : Sample bit en Master Mode =1 : les données en entrée sont échantillonnées en fin de cycle.
Bit CKE : Clock Edge Front actif de l'horloge : En I2C CKE=0.
Bit D/A : Drapeau indiquant si une adresse ou une donnée a été transmise.
Bit P : Stop Bit Si P =1 détection d'un stop.
Bit S : Start bit Si S =1 indique qu'un bit de start a été détecté.
Bit R/W : bit Read/Write
Bit UA : update Adress : utilise en mode 10 bits uniquement.
Bit BF : Buffer Full Status Bit :
En réception BF =1 quand le registre de réception est plein.
En émission BF =0 quand le registre de transmission est vide.
Ce bit s'appelle STAT_BF dans le programme C.

Programme :
SSPSTAT = 0x80; // Bit SMP=1, en I2C mode 100Khz.

Le Registre SSPCON (Registre de contrôle)

Bit SSPEN : Synchronous Serial Port Enable En Mode I2C SSPEN =1 pour valider les lignes SDA et SCL sur RC3 et RC4.
Bit CKP : Clock Polarity Select Avec CKP =1 en mode esclave pour valider l'horloge.
Bits SSPM3:SSPM0 : Selection du Mode de Fonctionnement :
Ici :1000 en Master Mode.

Programme :
SSPCON = 0x38; // SSPM0 à 3 mode I2C Maître, SSPEN=1 sorties sur RC3 et RC4, CKP=1 validation horloge.

Le Registre SSPCON2 (Registre de contrôle N°2)

Bit GCEN : Utilisé en mode Esclave.
Bit ACKSTAT : Acknowledge Status. Ce bit =1 lorsque le Maître reçoit un ACK.
Bit ACKDT: Bit positionné à 0 par le Maître lors d'une lecture pour acquitter la donnée.
Bit ACKEN : En Master Mode bit de validation d'un acquittement avec ACKEN =1.
Bit PEN : Stop Bit Si PEN =1, envoi d'un Stop par le Maître.
Bit RCEN : En réception du Maître RCEN =1, pour orienter SDA en entrée de lecture.
Bit SEN : Start bit Si SEN =1, envoi d'un Start par le Maître.
Bit RSEN : Repeat Start, envoi d'un Repeat Start par le Maître.

Programme :
SSPCON2 = 0x00; // mode maître attente acquittement esclave.

Le Registre SSPADD :

Ce registre contient une valeur sur 8 bits permettant de calculer la fréquence de l'horloge Fclock.
Pour nos applications Fclock = 100Khz.
Application :
Fclock = Fosc / 4(SSPADD + 1) = 100 Khz. (avec Fosc = 4 Mhz).

Programme :

```
SSPADD  = 9;                    // Pour avoir Fclock= 100Khz.
```

Le Registre SSPBUFF :

Registre Buffer en transmission de données en émission et en réception

Programme :

```
rejet   = SSPBUF;               // vide le buffer.
```

Le Bit SSPIF du Registre PIR1 :

SSP Interrupt Flag. En mode Master ce Flag permet de tester les fins de séquences Start, Stop, Restart et les acquittements.

5) Expérimentations :

Nous utilisons deux circuits différents en communication I2C :

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

Le but de ces manipulations est de se familiariser avec le bus I2C.
Nous utilisons un analyseur logique pour l'observation des états physiques des composants mis en jeu.
Cet appareil 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 d'accès au bus I2C.
Cela permet de trouver rapidement les origines des dysfonctionnements :

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

5-1 Le convertisseur de température :

5-1-1 Fonctionnement :

Le 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

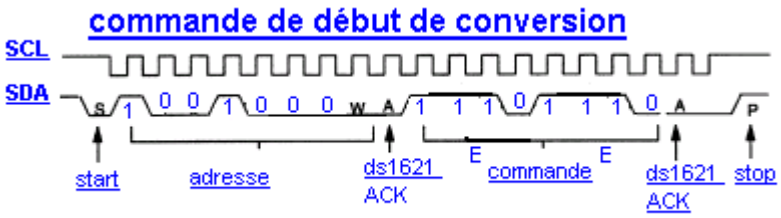
choix de l'utilisateur. (Possibilité d'utiliser jusqu'à huit capteurs à des adresses différentes).

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

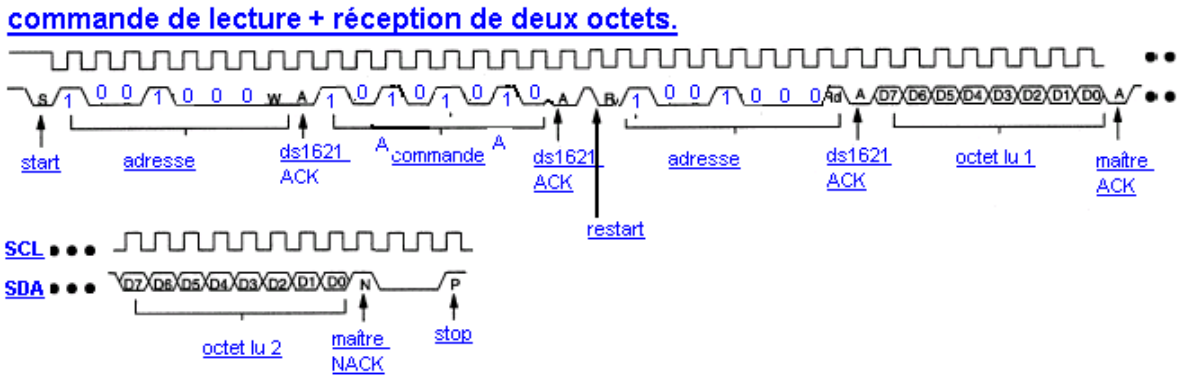
Relation entre la valeur numérique et la température

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

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



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).

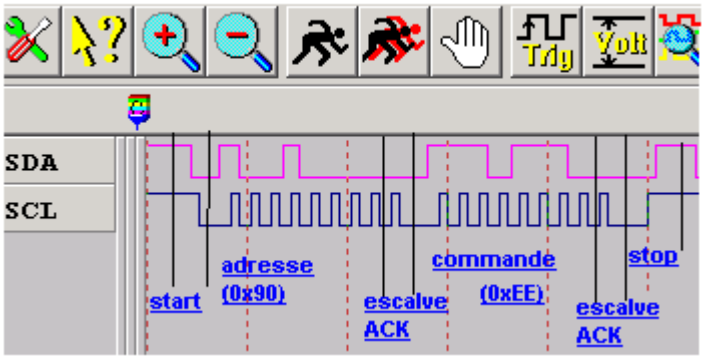


5-1-2 Manipulation :

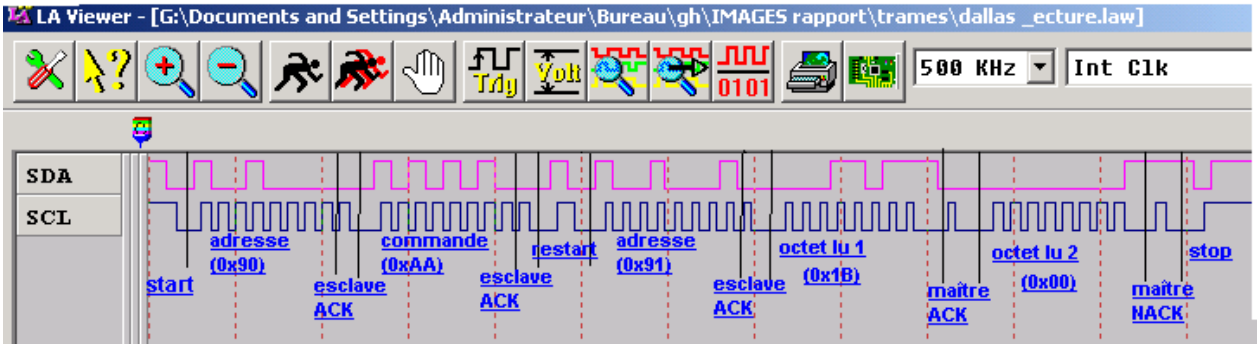
Une fois le capteur connecté correctement (la broche SDA du capteur relié à la broche

SDA du PIC...) et les registres configurés, il suffit de programmer les trames à envoyer au capteur selon le format donné ci-dessus.
En se connectant aux lignes SDA et SCL avec l'analyseur logique, on peut observer l'état physique des lignes et donc 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



Lecture des valeurs de température 0x18 et 0x00 : soit 25 degrés C

5-2 La mémoire EEPROM 24lc256

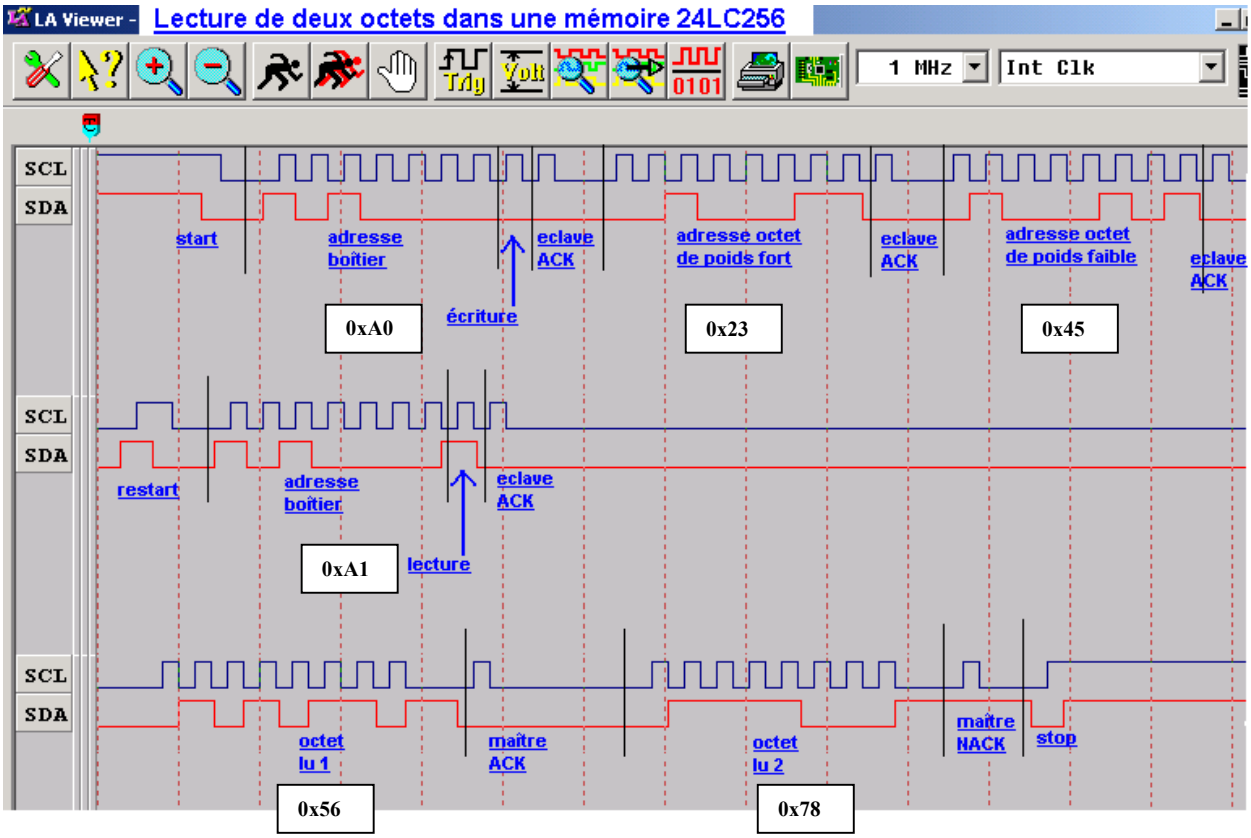
5-2-1 Fonctionnement

La mémoire 24lc256 répond aux commandes suivantes :
-L'adresse du boîtier est codée sur 7 bits.
-L'adresse octet est codée sur 14 bits (donc envoyée sur deux octets).

5-2-2 Manipulation :

Nous demandons la lecture de deux octets de données situés dans les cases mémoires d'adresse 0x2345 et 0x2346, et ce dans le boîtier répondant à l'adresse : 0xA0.
Voici la trame obtenue à l'aide de l'analyseur logique.

Demande de lecture de deux octets à ces adresses, on obtient la trame ci-dessous :



Les valeurs lues sont 0x56 et 0x78.

6- 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.
Nous travaillons en mode mono-maître, nous n'avons pas à nous préoccuper de la gestion des conflits.
En revanche le protocole ne prévoit aucune vérification des données transmises. Il est alors difficile de savoir si les valeurs réceptionnées sont exactes.
Les réponses des composants sont parfois un peu tardives ; il faut donc ajuster les délais d'attente en réception.

7- Programme I2C (Exemple)

```
/* Définition des bibliothèques*/
# include <pic.h>
# include "biosdem.h"
/* configuration */
  __CONFIG(0x3539);
/* Déclaration des fonctions */
void LC_read (char,char,char,char);          // fonction de lecture I2C de n octets.
void LC_write_un (char,char,char,char);      // fonction d'écriture I2C d'un octet.
void LC_write_plus (char,char,char,char);    // fonction d'écriture I2C de n octets.
void init (void);                            // fonction d'initialisation du circuit I2C.

/* Déclaration des variables */
Unsigned char i, rejet;
Unsigned char adb= 0xA0, adoh= 0x10, adol= 0x00, nb0= 10, ml[10], mae[10];

/* Programme principal */
void main (void)
{
        init();
        for( ; ; )
        {
        LC_write_un (adb, adoh, adol, mae[0]);
        LC_write_plus (adb, adoh, adol, nbo);
        LC_read (adb, adoh, adol, nbo);
        {
{


/************************************************************/
/*  Déclaration de la Fonction                              */
/*       Nom de la Fonction :        void init (void)       */
/*  Description: Cette fonction initialise le contrôleur I2C du PIC */
/*       Paramètres d'Entrée :   Aucun                      */
/*       Paramètre de Sortie :    Aucun                     */
/************************************************************/
void init (void)
{
        TRISC3  = 0;                //Ligne horloge SCL en sortie.
        TRISC4  = 1;                //Ligne données SDA en entrée.
        SSPSTAT = 0x80;             // Bit SMP=1, en I2C mode 100Khz.
        SSPCON  = 0x38;             // SSPM0 à 3 mode I2C Maître, SSPEN=1 sorties sur RC3
et RC4,  CKP=1 validation horloge.
        SSPCON2 = 0x00;             // mode maître attente acquittement esclave.
        SSPADD  = 9;                // Pour avoir Fclock= 100Khz.
        rejet   = SSPBUF;           // vide le buffer.

}
```

```
/*********************************************************************/
/*  Déclaration de la Fonction                                       */
/*       Nom de la Fonction : void LC_read (char,char,char,char)     */
/*  Description: Cette fonction lit n octets à partir d'une adresse boîtier  et */
/*       d'une adresse mémoire spécifiée sur deux octets             */
/*       Paramètres d'Entrée :  b= @boîtier, oh=@haute mem,  ol= @basse mem */
/*                              n= nombre d'octets.                  */
/*       Paramètre de Sortie :  Aucun.                               */
/*********************************************************************/
void LC_read (char b, char oh, char ol, char n)
{       int i:
                SEN   = 1;                  //Start
                while(SEN);                 //attente fin de start
                SSPIF = 0;
                SSPBUF= (b & 0xFE);         //@boîtier en écriture
                while(!SSPIF);              // test flag ACK
                SSPIF = 0;
                SSPBUF= oh;                 //@ mémoire haute
                while(!SSPIF);              // test flag ACK
                SSPIF = 0;
                SSPBUF= ol;                 //@ mémoire basse
                while(!SSPIF);              // test flag ACK
                SSPIF = 0;
                RSEN  = 1;                  //Restart
                while(RSEN);
                SSPIF = 0;
                SSPBUF= (b | 0x01);         //@boîtier en lecture.
                while(!SSPIF);              // test flag ACK
                for(t=0;t<100;t++)
                { }
        for(i=0; i<(n-1); i++)          // Remplissage tableau ml[i]
        {
                RCEN  = 1;                  // Validation SDA en lecture
                while(STAT_BF == 0);
                ml[i] = SSPBUF;             // lecture du I ième élément dans ml[i]
                ACKDT = 0;                  // Acquittement du maître
                ACKEN = 1;
                while(ACKEN == 1);
                SSPIF = 0;
        }
        RCEN  = 1;                          // Validation SDA en lecture
        while(STAT_BF == 0);
        ml[(n-1)] = SSPBUF;                 // lecture du dernier élément dans ml[i]
        ACKDT = 1;                          // Non  Acquittement du maître
        ACKEN = 1;
        while(ACKEN == 1);
        SSPIF = 0;
        PEN   = 1;                          //Stop
        while(PEN == 1);
        DelayMs(250);                       //Tempo de fin de trame
}
```

```
/*********************************************************************/
/*   Déclaration de la Fonction                                      */
/*        Nom de la Fonction : void LC_write_un (char,char,char,char) */
/*   Description: Cette fonction écrit un octet à partir d'une adresse boîtier   et */
/*       d'une adresse mémoire spécifiée sur deux octets             */
/*        Paramètres d'Entrée :  bt= @boîtier, oth=@haute mem,  otl= @basse mem */
/*                               mt= valeur de l'octet à écrire.      */
/*        Paramètre de Sortie :  Aucun.                               */
/*********************************************************************/

void LC_write_un (char bt,char oth,char otl,char mt)
{
        SEN   = 1;                  //Start
        while (SEN == 1);           //attente fin de start
        SSPIF = 0;
        SSPBUF= (bt & 0xFE);        //@boîtier en écriture
        while (SSPIF == 0);
        SSPIF = 0;
        SSPBUF= oth;                //@ mémoire haute
        while (SSPIF == 0);         // test flag ACK
        SSPIF = 0;
        SSPBUF= otl;                //@ mémoire basse
        while (SSPIF == 0);         // test flag ACK
        SSPIF = 0;
        SSPBUF= mt;                 Ecriture de la valeur en mémoire EEPROM.
        while (SSPIF == 0);         // test flag ACK
        SSPIF = 0;
        PEN   =1;                   //Stop
        while(PEN);
}
```

```
/*********************************************************************/
/*   Déclaration de la Fonction                                      */
/*        Nom de la Fonction : void LC_write_plus(char,char,char,char) */
/*   Description: Cette fonction écrit n octets à partir d'une adresse boîtier   et */
/*       d'une adresse mémoire spécifiée sur deux octets             */
/*        Paramètres d'Entrée :  boi= @boîtier, octh=@haute mem,  octl= @basse mem */
/*                               nb= nombre d'octets à écrire.        */
/*        Paramètre de Sortie :  Aucun.                               */
/*********************************************************************/

void LC_write_plus (char boi,char octh,char octl,char nb)
{
int i;
        SEN   = 1;                  //Start
        while (SEN);                //attente fin de start
        SSPIF = 0;
        SSPBUF= (boi & 0xFE);       //@boîtier en écriture
        while (!SSPIF);             // test flag ACK
        SSPIF = 0;
        SSPBUF= octh;               //@ mémoire haute
        while (!SSPIF);             // test flag ACK
        SSPIF = 0;
        SSPBUF= octl;               //@ mémoire basse
        while (!SSPIF);             // test flag ACK
        for(i=0; i<nb; i++)         // Ecriture multiple en mémoire EEPROM.

        {
                SSPIF = 0;
                SSPBUF= mae[i];     // écriture du I ième élément dans EEPROM
                while (!SSPIF);     // test flag ACK
        }
        SSPIF = 0;
        PEN   =1;                   //Stop
        while(PEN);
        DelayMs(5);                 /*/Tempo de fin de trame.
}
```