Université de Franche-Comté

UFR Sciences et techniques

MASTER M2 M2E2-ESE

Année 2015

Euphrasie Sebastien

Systèmes Embarquées

Thème :

I2C entre MC9S12DT256 et DS1621

Étudiants :

TORRES ZETINO Juan Carlos

MOUSSA MIGUIL Rachid

Besançon le 29 Octobre 2015

Table de Matière

Introduction

Dans la première partie du TP nous avons étudié la documentation du DS1621 et du bloc I2C du microcontrôleur MC9S12 afin de repérer les registres de configuration et déterminer les valeurs à leur donner.

Ensuite nous proposons un montage pour faire fonctionner le DS1621 avec le microcontrôleur ainsi qu'un organigramme qui décrit les différentes étapes à affranchir pour garantir la communication de données à travers le bus I2C

Dans la deuxième partie nous avons réalisé un programme permettant au microcontrôleur de pouvoir, après configuration du thermomètre, lire dans une boucle infinie la température ambiante et l'afficher en binaire sur les 4 LEDs (précision au 1°C).

Dans la partie final nous allons réaliser un test de vérification Analogique en analysant les signaux de commande envoyées à travers de l'oscilloscope pour garantir la fiabilité de notre système

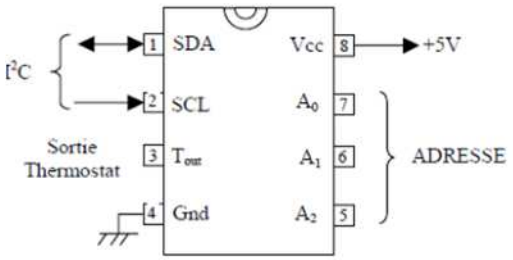
1 Analyse de documentation DS1621 et module I2C du MC9S12

Lire et analyser la documentation du bloc I2C du microcontrôleur MC9S12 et DS1621 En particulier, repérer les registres de configuration et déterminer les valeurs à leur donner.

Solution :

1.1 Capteur de température DS1621

En consultant la documentation du DS1621, nous identifions les différentes fonctionnalités des broches du capteur.



SDA	- 2-Wire Serial Data Input/Output
SCL	- 2-Wire Serial Clock
GND	- Ground
T_{OUT}	- Thermostat Output Signal
A0	- Chip Address Input
A1	- Chip Address Input
A2	- Chip Address Input
V_{DD}	- Power Supply Voltage

Fig1 : brocher du capteur DS1621

Il dispose de 8 broches et la communication i2c peut être réalise à 100khz ou 400khz selon le choix.

Fonctionnement :

En consultant la documentation du DS1621, nous remarquons les commandes suivantes :

COMMANDE(en hexa)	DESCRIPTION
AC	Accès au registre de configuration (lecture ou écriture).
EE	Début de conversion.
22	Arrêt conversion.
AA	Lecture de la température. DS1621 renvoi 2 octets.
A1	Lecture ou écriture du seuil haut du thermostat : TH.
A2	Lecture ou écriture du seuil bas du thermostat : TL.

Fig2 : tableau de commandes utilisé pour interagir avec le DS1621

Les commandes AC, EE et 22 ne sont pas suivie de l'envoi de données.
Par contre la commande AA est suivie de l'envoi par le DS1621 de 2 octets
Représentant la température, dans notre programme étant donné que nous utilisons le mode ONE SHOT, la requête de conversion 0xEE sera effectuée tous les cycles d'exécution.
Les commandes A1 et A2 sont suivies elles aussi de 2
Octets envoyés par le maître dans le cas d'une écriture des seuils du thermostat ou bien retournés par le DS1621 dans le cas d'une lecture.

Tout dialogue du Maître avec un DS1621 esclave doit débuter par l'envoi d'un mot de CONTROLE.

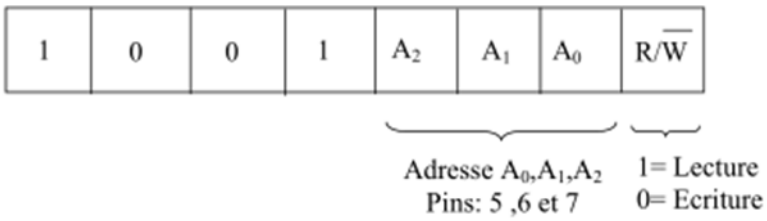


Fig3: registre de contrôle DS1621

1.2 Bloc I2C du microcontrôleur MC9S12

Repérer les registres de configuration et déterminer les valeurs à leur donner.

Solution :

Nous nous disposons à analyser les diffèrent registres à utiliser pour configurer le bus I2C sur le microcontrôleur ,dans notre cas il y existe 5 registres qui permettent entre autres de gérer les signaux de START et STOP , la fréquence de transmission et les statuts de réception de données

Les registres à configurer sont les suivantes : IBCR, IBAD, IBFD, IBSR, IBDR

Registre IBCR:

C'est un registre qui permet de configurer l'interface I2C pour assurer transmission et réception de l'information, il est structuré de la manière suivant :

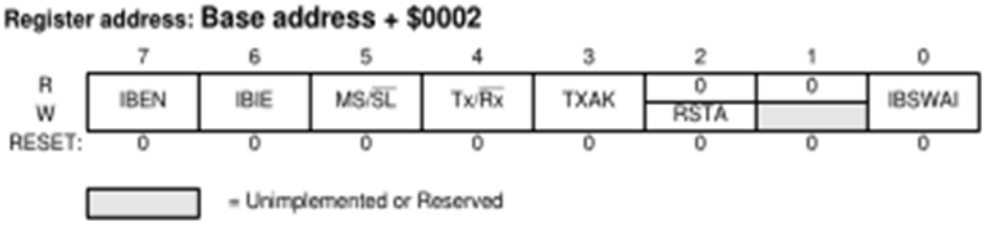


Figure 3-4 IIC-Bus Control Register (IBCR)

Dans notre cas nous utilisons les bits suivants :

- IBEN : permet d'habiliter l'interface I2C dans le microcontrôleur
- MS/SL : permet de générer le signal de START de communication
- TX/RX : permet de configurer le Maitre en mode Transmissions ou Réception de données.
- TXAK : génère une Acknoledge par défaut quand le maitre est en mode réception.
- RSTA : permet de générer un repeated-START .

Registre IBAD:

Permet d'ajouter une adresse d'identification de Maitre dans le cas qu'il soit interrogé pour une autre diapositive connectée au bus I2C de manière active.

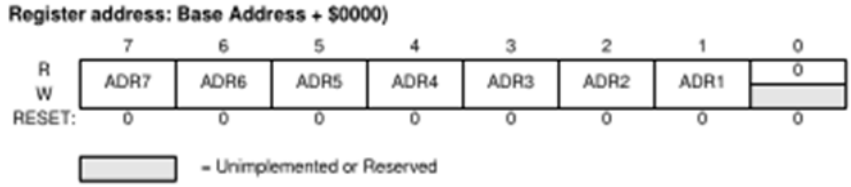


Figure 3-1 IIC Bus Address Register (IBAD)

Registre IBFD

Nous utilisons ce registre pour configurer la fréquence de transmission du Bus I2C.

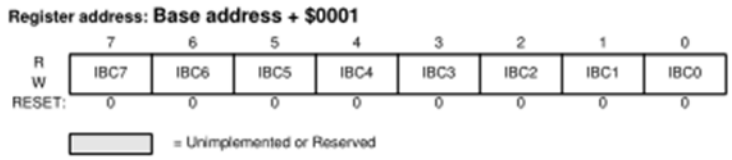


Figure 3-2 IIC Bus Frequency Divider Register (IBFD)

Registre IBSR

Ce registre permet d'obtenir information sur l'état de la transmission de données sur le bus I2C, nous utiliserons les bits suivants :

- IBIF : C'est une bit qui sera mis à « 1 » lors de la transmission de données soit fini dans le bus.
- RXAK : c'est une bit qui indique si l'esclave a bien reçu tous les Bytes

Figure 3-5 IIC Bus Status Register (IBSR)

Registre IBDR

Ce registre a deux modes de fonctionnement, quand le Maitre est configuré comme Transmetteur , il permet d'envoyer les données sur le bus I2C dès qu'on écrit sur ce registre, par contre quand il est configurée comme Maitre récepteur, il récupère les valeurs de bits envoyées par l'esclave en lisant sur IBDR

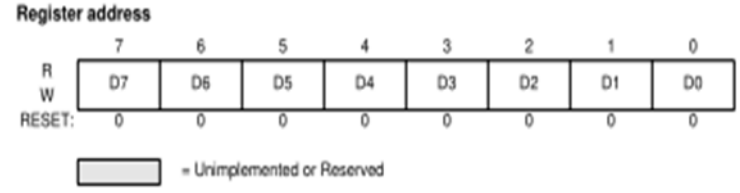


Figure 3-6 IIC Bus Data I/O Register (IBDR)

1.3 Sélection de Prescalar

Pour choisir le prescale qui permet la configuration de la fréquence de transmission à 100Kbps du bus I2C nous réalisons l'opération suivante

Soit la fréquence de la carte de 4MHZ, nous devrons diviser par 2, 2MHZ pour obtenir son cycle de travail, ensuite nous calculons son preescalar ainsi :

2MHZ/100KHZ=**20**

Cette valeur est configurée grâce au registre IBFD en mettant tous ses bit à Zéro
IBFD =0X00

2 Montage et identification de Pins pour communication Bus I2C

Proposer un montage pour faire fonctionner le DS1621 (en particulier les connexions avec le microcontrôleur et tous les composants nécessaires en plus de la puce DS1621 et de la carte microcontrôleur)

Solution :

Afin de connecter le microcontrôleur MC9S12 avec le thermomètre numérique DS1621 nous devons prévoir les points suivants
SDA : câble de données entre Maitre et esclave
SCK : câble de signal d'horloge pour synchroniser la communication entre le Maitre et l'esclave.
2x Résistance de Pull Up 4.7Kohms : ils permettent de maintenir le niveau en logique 1 dans le câbles SDA et SCK c.à.d. quand les sorties se trouvent inactives aucun dispositif utilise les lignes, ils auront un état de 5V, le maitre et l'esclave aurons droits uniquement à baisser le niveau à Zéro

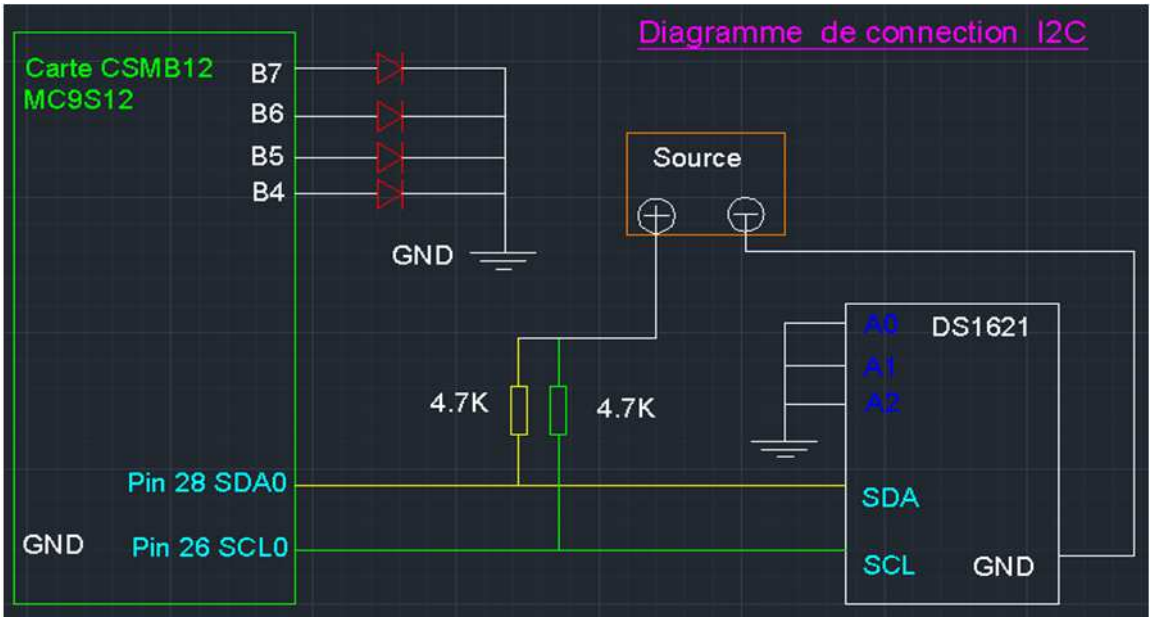


Fig4 : Schéma de connexion Carte CSMB12

Pour implémenter cette configuration nous utilisons la carte en connexion **J1** qui habilite le module de I/O Ports dans lequel se trouve les deux fils nécessaires pour la communication I2C dans notre cas cela sera PIN 26 et PIN 28.

PA7/***ADDR15/DATA15***	25	26	PJ7/KWJ7/TXCAN4/SCL0
PA6/***ADDR14/DATA14***	27	28	PJ6/KWJ6/RXCAN4/SDA0

Distribution de pins sur le port I/0 carte CSMB12

Finalement pour représenter la température, il faudra configurer les 4 LEDS sur la carte CSMB12 dans le PORTB en configuration active LOW.

LED1	PB4/***ADDR4/DATA4***	Green LED (LED1)
LED2	PB5/***ADDR5/DATA5***	Green LED (LED2)
LED3	PB6/***ADDR6/DATA6***	Green LED (LED3)
LED4	PB7/***ADDR7/DATA7***	Green LED (LED4)

Distribution de pins sur le port utilisateur I/0 carte CSMB12

2.1 Sélection de plage de température pour l'Affichage

Donner la plage de température choisie pour l'affichage

Solution

Dans notre système le DS1621 envoie **2 Octets de données** après requête de température , nous utiliserons seulement l'octet le plus fort , car nous voulons une précision de 1°C sans prendre en compte le décimaux ,ainsi pour pouvoir afficher la réponse nous utiliserons 4 LEDs , c.à.d. que nous aurons 16 valeurs maximum à afficher , nous choisissons une plage mesurable par rapport à la température de la salle 16-31°C

Plage de température : 16-21°C

- **0x10=0001 0000 = 16°C limite inferieur**
- **0x1F=000111111=31° C limite Supérieur**

Avec LED4, LED3, LED2 et LED1 en logique inverse comme interface, ils seront programmés à travers le DDRB registre PORTB.

3 Organigramme de système de mesure de température

Dessiner un organigramme du système :

Solution :

A continuation nous présentons l'Organigramme de notre système de mesure qui permet de connaitre les différentes étapes à parcourir à fin de réaliser une communication Maitre-Esclave entre le microcontrôleur et le DS1621.

Les étapes seront les suivantes

Dans la première partie le maitre devra être configuré sur le microcontrôleur il sera à charge de gère la communication à travers de l'interface I2C, il donne les impulsions d'horloge SCK qui permettent la transmission ou réception de données à 100kbps et il génère les signaux de STOP et START selon le type de commande à implémenter.

Dans un deuxième partie L'esclave sera le capteur de température DS1621 , qui reçois les instruction provenant du bus I2C, il devra faire la conversion de température selon le mode de fonctionnement choisi dans le registre de contrôle continu ou One Shot

Finalement le **DS1621 envoie les données de température** vers le microcontrôleur, ensuite l'affichage se réalise en utilisant les 4 leds en Sortie de la carte d'expérimentation

Organigramme Système de mesure de température

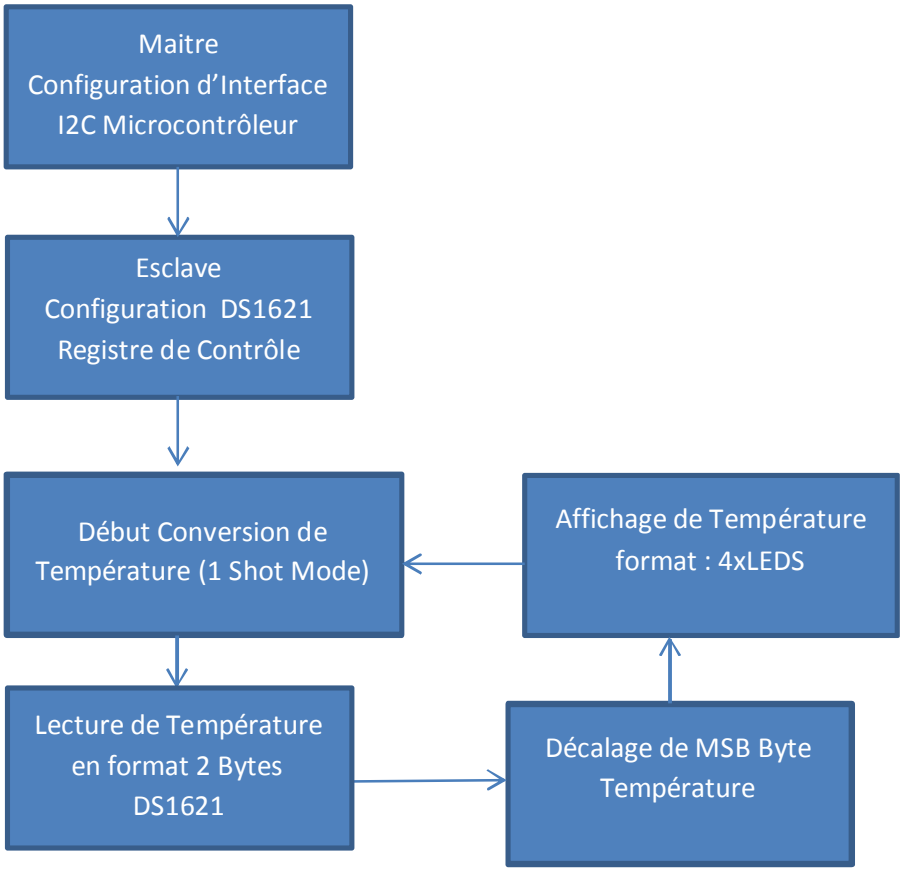


Fig5 : Organigramme du système de mesure en utilisant le modèle de transmission Master- esclave

4 Réalisation du Programme de mesure et analyse de fonctionnes

Réaliser un programme permettant au microcontrôleur de pouvoir, après configuration du thermomètre, lire dans une boucle infinie la température ambiante et l'afficher en binaire sur les 4 LEDs (précision au °C) .

Solution

Notre programme consiste en générer une système de lecture de température indépendant de l'esclave ou capteur à connecter , l'approche à utiliser sera de créer une ensemble de sous-fonctionnes génériques qui permettront d'envoyer les commandes nécessaires pour implémenter l'organigramme , par exemple nous devrons coder des fonctionnes pour écrire 1 Byte sur le esclave , récupérer les données en lecture afficher les données , etc. Dans la suite nous expliquons le programme en détaille

4.1 Description de Fonctionnes

Afin de mettre en place la communication entre le Maitre et le capteur DS1621 à travers du bus I2C il faut spécifier les fonctionnes suivantes :

```
//*********************************************************************//
//MACROS Registre de Config  IBCR
#define IBEN 7 //Bit 7     1=enable I2C
#define MS   5 //Bit 5     1=Master enable START generated
#define TX   4 //Bit 4     1=TX/RX Master Transmit enable
#define TXAK 3 //Bit 3     1=No ACK sent apres horloge 9
#define RSTA 2 //Bit 2     1=Repeated START sent
//GLOBAL VARIABLES
char tempH;    //MSB byte envoyée depuis DS1621
char tempL;    //LSB byte énvoyée depuis DS1621
char dummy;              //variable vide pour stocker valeur Temp FAUX
//FONCTIONS PROTOTYPES
void init_i2c(char adresse);
void envoyer_commande(char adresse, char commande);
void ecrire_byte(char adresse,char commande,char data);
void lecture2byte(char adresse_w,char adresse_r,char commande);
void attente_i2c_a(void);
void attente_i2c_b(void);
void wait(int mili ,int itera) ;
void affiche(char tempp);
//*********************************************************************//
```

Dans la première partie de notre programme nous définissons le MACROS pour identifier les différents bits du registre de configuration **IBCR**

Ensuite nous allons définir 2 variables globales **tempH** et **tempL** qui vont stocker les valeurs de température, envoyé par le **DS1621**

Finalement nous définissons les fonctions suivantes pour interagir avec le DS1621 soit en écriture ou bien en lecture :

- **void envoyer_commande (char adresse, char commande)** :permet d'envoyer les commandes en 8 bits à travers du **Bus I2C** vers l'adresse

spécifié comme paramètre , cette fonction sera utilisé pour envoyer le commande **0xEE** qui indique le début de conversion de température .

- **void ecrire_byte(char adresse,char commande,char data) :** permet d'écrire une byte de données depuis le Maitre vers l'adresse spécifié comme paramètre , cette fonction sera utilisé afin d'envoyer le commande **0XAC** qui demande d'accéder au Registre de Contrôle du DS1621, après dans une deuxième temps nous allons envoyer la data à écrire sur ce registre pour configurer le mode **1 SHOT**
- **void lecture2byte(char adresse_w,char adresse_r,char commande) :**cette fonctionne permet de lire les 2 Bytes correspondant à la valeur de température mesuré par le DS1621 , Il faudra envoyer d'abord l'adresse en écriture suivi du commande lecture de température **0xAA** et spécifier l'adresse en lecture pour récupérer l'information
- **void attente_i2c_a(void) et void attente_i2c_b(void);** ces deux fonctionnes servent à attendre la finalisation de la transmission de données ,dans le premier type on attende que le bit ACK soit mis à zéro ainsi que le bit IBIF soit mis a 1 pour indiquer que la transmission est complet, la deuxième fonctionne d' attente est similaire mais on enlève la vérification par bit ACK car c'est le Maitre qui devra générer ce signal par défaut.
- **void affiche (char tempp)** : finalement la fonction affiche permet de réaliser une décalage de 4bits sur le MSB byte de température ,en autre nous changeons le PORTB associe aux leds 4,3,2 et 1pour représenter une plage de températures de 16°C .

4.2 Description de la fonction Main()

Notre fonctionne Main implémente l'organigramme dans un cycle de lecture infini, les instructions à utiliser pour le type de capteur choisi sont les suivantes

```
//*********************************************************************//
void main(void) {
//PARTIEA
DDRJ=0xC0;                    //Activation  Module I2C  sur carte CSMB12
DDRA=0x0F;                    //habiliter PORTA 4 Bits LSF comme OUTPUT
DDRB=0xF0;                     //habiliter PORTB en Sortie LEDS
PORTA=0x02;                    //Permet de génerer une boucle infinie while()
//PARTIEB
init_i2c(0x00);                //Config I2C sur  carte CSMB12
ecrire_byte(0x90,0xAC,0x01);//Config 1 SHOT Mode sur DS1621
//PARTIEC
while(PORTA==0x02)              //Mettre Bit1 PA1 PORTA comme 5V
{
envoyer_commande(0x90,0xEE); //Demmander debut de mesure SHOT
lecture2byte(0x90,0x91,0xAA);//Lecture   de 2 Bytes depuis DS1621
affiche(tempH);                //Affichage reponse en 4 LEDS
}
  EnableInterrupts;
  for(;;) {} /* wait forever */
  /* please make sure that you never leave this function */
}
```

Programme implémenté pour réaliser le système de mesure de température

Le programme main est compose des différents partie que nous allons expliquer dans la suite :

Partie A : Nous allons configurer le **registre DDRJ** en sortie pour activer les bits 7 et 6 **SDA et SCL** lors de la transmission de données selon les pulses d'horloge, ensuite **le registre DDRA** sera configuré en sortie dans le **bit 1** pour **obtenir 5 V** et alimenter le DS1621, finalement le registre **DDRB** permet de configurer les LEDS d'affichage de température liées au PORTB

Partie B : ensuite nous allons configurer les registres **IBFD, IBAD** et **IBCR** afin de initialiser l'interface I2C sur le microcontrôleur, ensuite nous allons configurer le DS1621 a travers de son registre de contrôle en choisissant le mode 1 SHOT qui permet de faire une seule capture de température à la fois.

Partie C : Dans cette partie nous envoyons le commande 0xEE vers le DS1621 pour demander le début de conversion de température, ensuite nous allons lire les 2 Bytes de données en envoyant le commande 0xAA, et finalement nous allons afficher la valeur de la température à travers de leds du Port B .

5 Test et Validation du Programme

Tester et déboguer votre programme avec les outils qui vous semblent nécessaire.

Solution :

Cette étape a comme finalité de tester le fonctionnement complète de notre système de mesure, pour ce faire nous allons dans un premier temps récupérer les valeurs logiques des leds affichées par le programme et vérifier que une augmentation de la température, fait changer l'état de leds .

La configuration finale utilise les résistances de pull up de 4.7 K ohms branchées aux fils SDA et SCL.

5.1 Vérification par affichage de LEDs

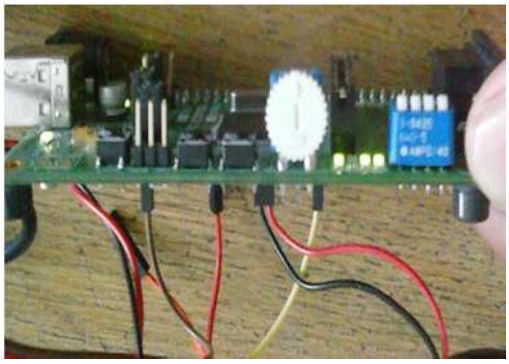


Fig6 : lecture d'état de leds 27°C

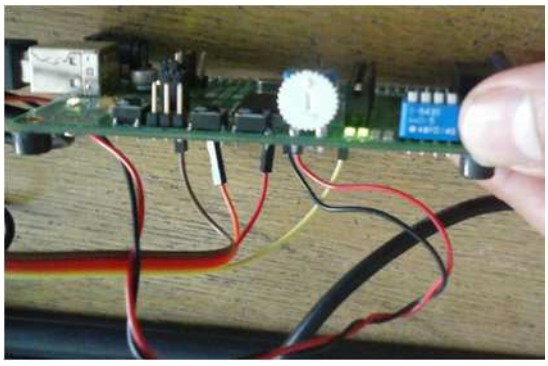


Fig7: lecture de l'état de led 28°C

Les valeurs de LEDS obtenu dans un premier moment ont été comparées avec la plage de température choisi préalablement :

Limite inferieur 0x10=0001 0000 =16°C **LED4=OFF-LED3=OFF-LED2=OFF-LED1=OFF**

Limite superieur 0x10=0001 0000 =31°C **LED4=ON-LED3=ON-LED2=ON-LED1=ON**

Dans notre cas nous avons obtenu une température initiale dedans la salle 215B le mardi 27 octobre 2015 à midi, la valeur de 27°C qui correspond aux états des leds suivantes :

Température de la Salle 0x10=0001 1011 =**27°C LED4=ON-LED3=OFF-LED2=ON-LED1=ON**

Ensuite nous avons réchauffé le capteur avec la chaleur interne provenant de la main, la température est montée jusqu'à la valeur suivante.

Température de la Salle2 0x10=0001 1011 =**28°C LED4=ON-LED3=ON-LED2=OFF-LED1=OFF**

Cette expérience montre que le leds affichent les variations de température mesurée par le DS1621 , nous avons décidé de récupérer le MSB byte car c'est le Byte qui correspond à la plage de température choisi , par ailleurs le décalage de 4 bits sert à visualiser en temps réel les 4 bits plus sensibles aux changement de 1°C.

5.2 Vérification par Oscilloscope

Afin de valider la qualité des signaux logiques générés par la carte CSMB12 lors de la communication I2C ,nous avons connecté les pins SDA et SCL à l'oscilloscope pour capturer le comportement des niveaux logiques lors de l'envoie du commande début de conversion fonction void **envoyer_commande (0x90, 0xEE)** dans notre programme :

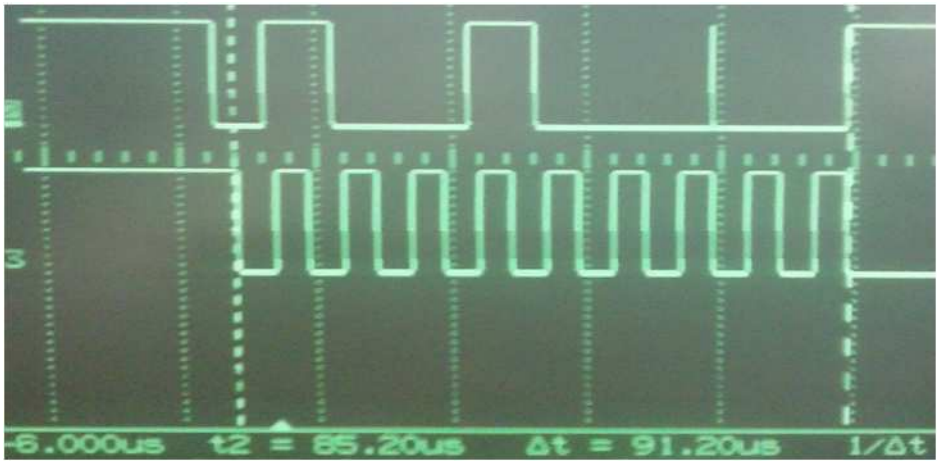


Fig8 : signal numérique 0x90 qui représente l'adresse du DS1621 en écriture

Partie A (Adresse en écriture 0x90)

La trame de données ci-dessus représente l'adresse en écriture **0X90** envoyée par notre programme pour pouvoir se connecter au capteur DS1621, la période de l'envoie de Byte a une durée de 91.20 us avec une pulse de horloge envoyé tous les 10 us que nous avons vérifié dans l'oscilloscope , cette valeur corresponde à une fréquence de transmission par bit de 100Khz qui corresponde à la vitesse par default du bus I2C utilisée .

Partie B (Commande 0xEE)

La conversion de température se fait en envoyant le commande **0xEE** depuis le Maitre vers le capteur DS1621, dans l'oscilloscope nous voyons le nombre binaire **1110 1110** qui est envoyé sur 8 pulses d'horloge avec le signal de ACK généré par le DS1621 et le signal de STOP généré par le Maitre ,nous voyons aussi la concordance entre les valeurs logique sur le bus **SDA** et les paramètres spécifie dans notre programme

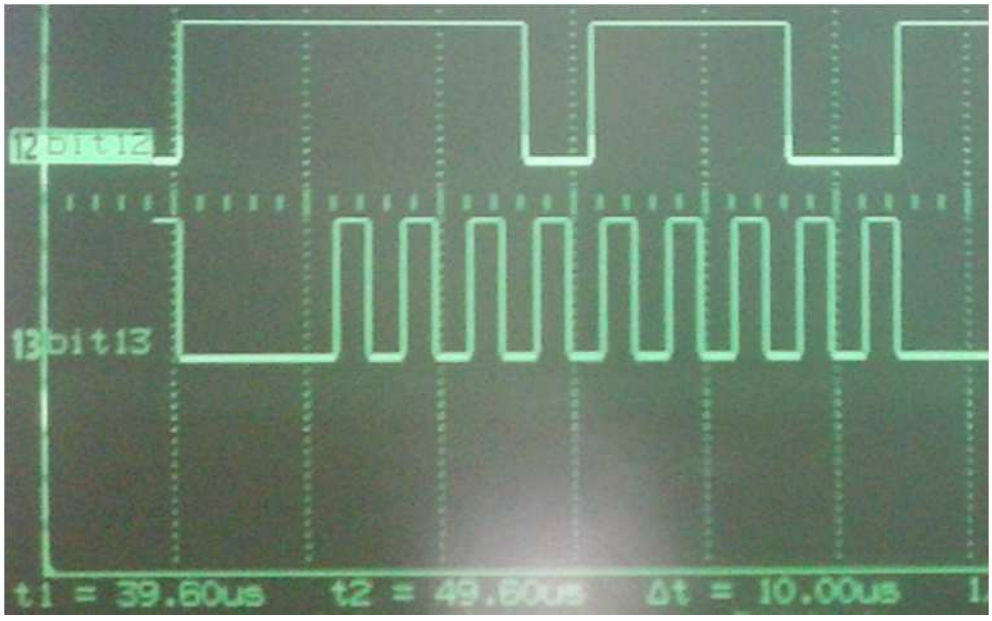


Fig9 : signal numérique 0XEE qui représente la commande de début de conversion

Conclusion

Le présent travail pratique nous a permis d'approfondir les connaissances dans la programmation du microcontrôleur mc9s12dt256, à travers de la carte de test CSMB12 notamment dans l'implémentation de l'interface I2C pour commander le capteur de température DS1621.

Nous avons obtenu une meilleur compréhension des différents registres liés au bus I2C et le rapport entre les diffèrent bits qui permettent de configurer les fréquences de transmission du bus I2C ainsi que l'analyse de statut des paquets envoyées.

Au terme de notre recherche, nous avons réalisé un programme permettant au microcontrôleur de lire la température ambiante et de l'afficher en binaire sur les quatre LEDs .

Nous avons pendant la réalisation de ce programme vu l'avantage de l'exploitation de la « data sheet». et ce quelle pouvait nous apporter dans notre programme.

Cela nous a permis d'en apprendre plus sur le fonctionnement du microcontrôleur MC9S1DT256 et du capteur de température DS1621.

Finalement nous avons appris l'importance d'avoir un organigramme complète du système avant de commencer le développement du software, cela permet de savoir quelles sont les fonctionnes à programmer ainsi que la structure du boucle de lecture de température pour pouvoir faire une application avec le moins de ressources possibles.

6 Annexe

6.1 Programme de mesure

Dans la suite nous présentons le programme réalisé en l'IDE Code Warrior qui a permis la supervision de la température de la salle 215B.

```
#include <hidef.h>      /* common defines and macros */
#include <mc9s12dt256.h>     /* derivative information */
//*************************************************************//
//MACROS Registre de Config  IBCR
#define IBEN 7 //Bit 7    1=enable I2C
#define MS   5 //Bit 5    1=Master enable START generated
#define TX   4 //Bit 4    1=TX/RX Master Transmit enable
#define TXAK 3 //Bit 3    1=No ACK sent apres horloge 9
#define RSTA 2 //Bit 2    1=Repeated START sent
//GLOBAL VARIABLES
char tempH;   //MSB byte envoyée depuis DS1621
char tempL;   //LSB byte énvoyée depuis DS1621
char dummy;           //variable vide pour stocker valeur Temp FAUX
//FONCTIONS PROTOTYPES
void init_i2c(char adresse);
void envoyer_commande(char adresse, char commande);
void ecrire_byte(char adresse,char commande,char data);
void lecture2byte(char adresse_w,char adresse_r,char commande);
void attente_i2c_a(void);
void attente_i2c_b(void);
void wait(int mili ,int itera) ;
void affiche(char tempp);
//*************************************************************//
void main(void) {
//PARTIEA
DDRJ=0xC0;               //Activation  Module I2C  sur carte CSMB12
DDRA=0x0F;               //habiliter PORTA 4 Bits LSF comme OUTPUT
DDRB=0xF0;                //habiliter PORTB en Sortie LEDS
PORTA=0x02;               //Permet de génerer une boucle infinie while()
//PARTIEB
init_i2c(0x00);           //Config I2C sur  carte CSMB12
ecrire_byte(0x90,0xAC,0x01);//Config 1 SHOT Mode sur DS1621
//PARTIEC
while(PORTA==0x02)          //Mettre Bit1 PA1 PORTA comme 5V
{
envoyer_commande(0x90,0xEE); //Demmander debut de mesure SHOT
lecture2byte(0x90,0x91,0xAA);//Lecture   de 2 Bytes depuis DS1621
affiche(tempH);              //Affichage reponse en 4 LEDS
}
  EnableInterrupts;
  for(;;) {} /* wait forever */
  /* please make sure that you never leave this function */
}
//FUNCTIONS Implementation
//*************************************************************
void init_i2c(char adresse)
{
IBFD=0x00; //config Prescalar Horloge SCL 2MHZ/100KHZ=20  Reg IBFD
IBAD=adresse;
IBCR=0x80;//Bit 7 IBEN  reg IBCR  activer
}
//*************************************************************
void envoyer_commande(char adresse ,char commande)
{
IBCR &=~(1<<TXAK);      //Bit 3 TXAK validation  ACK de master
IBCR|=(1<<TX) ;       //Bit 4 TX enable MASTER Transmis ON
IBCR|=(1<<MS) ;       //Bit 5 START envoyée
```

```
//Ecriture Adresse +Commande
IBDR=adresse;        //Addresse en ecriture du DS1621
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBDR=commande;         //ecrire sur le Bus un commande BYTE(8bits)
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBCR&=~(1<<MS)  ;    // envoyer  signal STOP
}
//************************************************************
void ecrire_byte(char adresse,char commande,char data)
{
IBCR &=~(1<<TXAK);      //Bit 3 TXAK validation  ACK de master
//IBCR|=(1<<TX) ;       //Bit 4 TX enable
//IBCR|=(1<<MS) ;       //Bit 5 START envoyée
IBCR|=0x30;
IBDR=adresse;         //Addresse en ecriture du DS1621
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBDR=commande;         //ecrire sur le Bus un commande
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBDR=data;             //ecrire sur le Bus un data
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBCR&=~(1<<MS);        // envoyer  signal STOP
}
//*********************************************************************
void lecture2byte(char adresse_w,char adresse_r,char commande)
{
IBCR &=~(1<<TXAK);      //Bit 3 TXAK validation  ACK de master
IBCR|=(1<<TX) ;//Bit 4 TX enable configurer Master pour trasmettre données
IBCR|=(1<<MS) ;//Bit 5 START envoyée
//Ecriture Write Adresse +Commande
IBDR=adresse_w;        //Addresse en ecriture du DS1621
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBDR=commande;         //ecrire sur le Bus un commande BYTE(8bits)
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
//Ecriture Lecture Addresse
IBCR |=(1<<RSTA);       //bit 2 RSTA Quand RSTA=1 generer "Repeated START"
IBDR=adresse_r ;       //Envoie Adresse en LECTURE vers DS1621
attente_i2c_a();     //Attente de ACK=0 BYTE  depuis  DS1621 + IBIF=1
IBCR &=~(1<<TX) ;       //Bit 4 TX=0 Master Receive Enable ,pour  Recevoir données depuis DS1621
dummy=IBDR;             // enregistrer la valeur precedent du registre IBDR
attente_i2c_b();       //Attente sans ACK d'esclave Master
//Lecture Exclusive de DS1621
tempH=IBDR;             //Lecture premier Byte envoyée par DS1621 et enregistrement dans variable temp
IBCR|=(1<<TXAK) ;      //bit 3 TXAK=1 generer un NOTACK apres Maitre ait recu BYTE
attente_i2c_b();       //Attente sans ACK d'esclave Master
IBCR&=~(1<<MS)  ;      // envoyer  signal STOP
tempL=IBDR ;           //Lecture LSB de la valeur  temperature depuis le DS1621
}
//*********************************************************************
void attente_i2c_a(void)
{
while(!(IBSR&0x02)) ;       //Bit 1 IBIF  mise à 1 indique Transfert Terminé
IBSR|=(IBSR<<1);            //Bit 1 IBIF  mise à 1 pour remise à ZERO
while(IBSR&0x01);           //Bit 0 wait until logique "0" is SET , la tranfert de Byte est terminée RXAK bit
}

//*********************************************************************
void attente_i2c_b(void)
{
while(!(IBSR&0x02)) ;       //Bit 1 IBIF  mise à 1 indique Transfert Terminé
IBSR|=(IBSR<<1);            //Bit 1 IBIF  mise à 1 pour remise à ZERO
}
//*********************************************************************
void wait(int mili ,int itera)
{
int i,k;
for(i=0;i<mili;i++)
```

```
{
for(k=0;k<itera;k++){}
}
}
//*********************************************************************
void affiche(char tempp)
{
char tempv=tempp ;
DDRB=0xF0;   //Configurer bits 7 6 5 4 MSB en "1" =OUTPUT pour activer les LEDS
tempv=~(tempv<<4);   //decalage de valeur de tempH=MSB vers le 4 bits le plus significatif du PORTB(LEDS)
PORTB=tempv;         //Allumage des LEDS logique Inverse
wait(100,4);       //Attendre 1 seconde avant continuer lecture suivante
}
//*********************************************************************
```