

# PILOTAGE DU BUS I2C À PARTIR DU MICROCONTRÔLEUR HCS12

Rachid MALTI<sup>1,2</sup>, Serge BOUTER<sup>2</sup>

<sup>1</sup>IMS UMR 5218 CNRS, Département LAPS – Université Bordeaux I  
351, cours de la Libération – F 33405 Talence cedex – France

<sup>2</sup> IUT Bordeaux I – Université Bordeaux I  
15, rue Naudet – CS 10207 – 33175 Gradignan Cedex – France.  
Tél : +33 (0)5 56 84 57 52

{malti, bouter}@iut.u-bordeaux1.fr

**Résumé**—L'objectif de cette communication est de présenter une solution matérielle et logicielle peu onéreuse pour la mise en place d'une maquette pédagogique permettant de piloter le bus I2C à partir du microcontrôleur HCS12. La solution proposée est basée sur l'utilisation d'un *starter kit* du commerce incluant un microcontrôleur de type HCS12, le développement d'un circuit intégrant deux composants I2C, et l'écriture de fonctions facilitant le pilotage du module I2C à partir du microcontrôleur. Cette maquette est utilisée depuis 2 ans pour la formation des étudiants de DUT et de la Licence Professionnelle Automatique et Informatique Industrielle spécialité Systèmes Automatisés et Réseaux Industriels du département GEII de l'IUT Bordeaux I.

**Mots-clés**— Bus I2C (ou IIC), microcontrôleur HCS12, circuit intégré, circuit imprimé, bus électronique.

## I. INTRODUCTION

Au début des années 1980, Philips Semiconductors développe un bus bidirectionnel à deux fils pour assurer une communication efficace *Entre circuits intégrés* ou en anglais *Inter-Integrated-Circuit* d'où l'acronyme IIC ou I2C. Actuellement, Philips commercialise plus de 150 Circuits Intégrés (CI) compatibles avec les spécificités du bus I2C, pouvant assurer une communication efficace entre des CI intelligents (e.g. microcontrôleurs), des CI généralistes (e.g. port d'E/S déporté, mémoire) et des CI orientés applications (e.g. traitement du signal, traitement audio et vidéo).

Le bus I2C a été développé pour des communications de très courte distance (de l'ordre du mètre) et pour un débit maximum de 100kbits/sec, comme le montre la fig.1. Une extension a par la suite été apportée, appelée FastI2C, permettant d'atteindre un débit de 400kbits/sec. Le principal intérêt du bus I2C est de simplifier la conception de circuits électroniques. Ainsi, les circuits de décodage d'adresses ne sont plus nécessaires et le routage des cartes électroniques se trouve sensiblement allégé, grâce à un système d'adressage constitué d'une adresse constructeur unique à chaque type de CI et d'une adresse locale permettant de distinguer des CI identiques sur la même carte. Actuellement, il existe plus d'un millier de CI compatibles I2C en provenance d'une cinquantaine de fabricants. L'I2C est devenu *de facto* un standard mondial.

## II. SPÉCIFICITÉS DU BUS I2C

Ce paragraphe rappelle les principes de fonctionnement du bus I2C. Le lecteur familier avec ces principes peut pas-

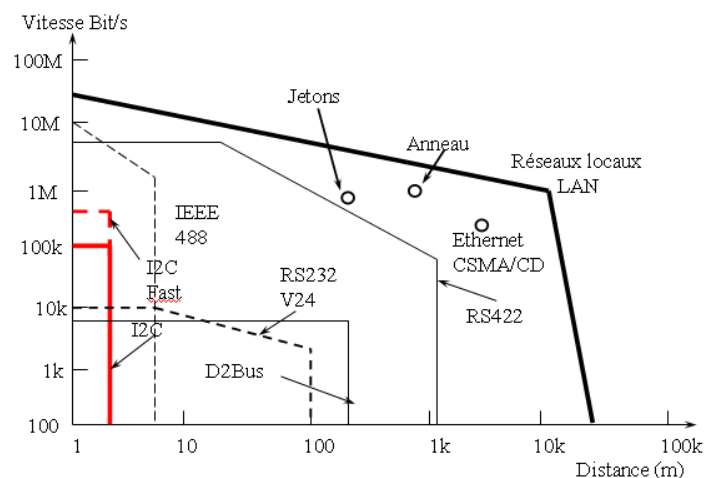


Fig. 1. Positionnement du bus I2C par rapport à d'autres bus et réseaux

ser directement à la section 3.

Bien que le bus I2C supporte des communications de type multi-maîtres multi-esclaves, il est piloté dans l'application développée par un maître unique, le problème d'évitement de collision ne se pose donc pas.

Le bus I2C s'articule autour de deux lignes de communication :

- Système DAta (SDA), ligne de transmission de données
- Système CLock (SCL), ligne d'horloge.

C'est toujours le maître qui génère l'horloge quel que soit le sens du transfert des données. Chaque ligne peut avoir deux états :

- '0', qui correspond à l'état dominant,
- '1', qui correspond à l'état récessif,

Comme le montre la fig2, il suffit qu'un nœud impose l'état '0' sur le bus, afin que celui-ci soit prédominant devant le '1'.

Tout changement d'état sur le bus I2C est permis sur l'état bas de l'horloge. Tous les nœuds du bus lisent l'état du bit sur l'état haut de l'horloge, conformément à la fig.3.

Les informations sont transmises sous la forme d'octets. Chaque octet est envoyé sur 9 bits, le 9ème bit servant

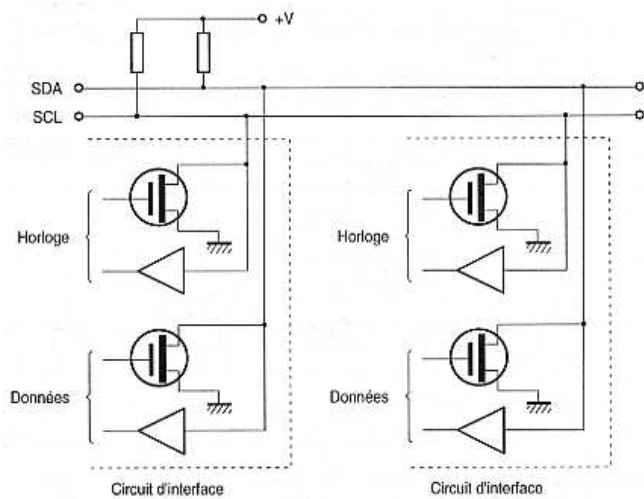


Fig. 2. Condition de départ et d'arrêt

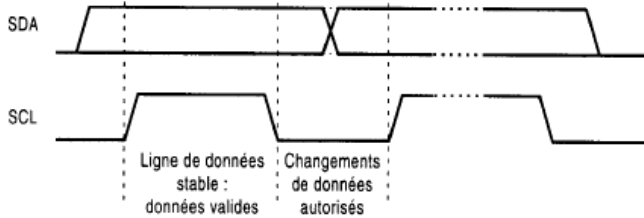


Fig. 3. Changement d'état et lecture sur le bus I2C

d'acquiescement (ACK). L'émetteur émet l'état logique '1' (état récessif) au 9ème bit et c'est le récepteur qui l'impose à '0' (état dominant) lorsqu'il reconnaît son adresse et après chaque octet reçu. C'est en vérifiant ce bit d'acquiescement que l'émetteur sait si la transmission se déroule correctement.

Au repos les lignes SDA et SCL sont à l'état haut. Une communication sur le bus débute par la génération de la condition de départ qui consiste à faire passer la ligne SDA à l'état bas alors que la ligne SCL est à l'état haut, conformément à la fig.4. La communication sur le bus I2C s'arrête par la génération de la condition d'arrêt où SDA passe à 1 alors que SCL est à 1. Les conditions de départ et d'arrêt sont imposées par le maître quel que soit le sens du transfert des données.

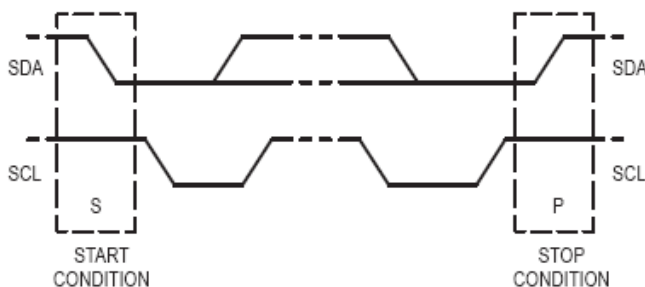


Fig. 4. Condition de départ et d'arrêt

Après avoir imposé la condition de départ, le maître en-

voie sur le bus l'adresse de l'esclave, codée sur 8 bit où le 8ème bit, bit de Lecture/Ecriture  $L/\overline{E}$ , indique le sens de transfert des données (du maître vers l'esclave ou vice-versa). L'adresse est généralement constituée de deux parties comme le montre l'exemple de la fig.5 : une adresse constructeur, unique à chaque type de CI et une adresse locale permettant d'utiliser plusieurs CI identiques sur une même carte.

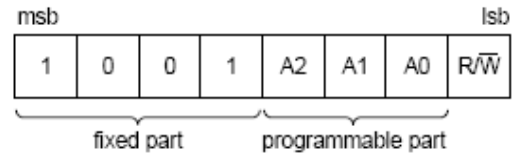


Fig. 5. Adresse du PCF8591

Il s'ensuit une ou plusieurs trames de données conformément à l'exemple de la fig.6.

### III. MODULE I2C DU HCS12

La conduite du bus I2C à partir du HCS12 se fait à partir d'un module dédié composé de 5 registres de 8 bits. Seuls les bits nécessaires à l'écriture des algorithmes du paragraphe IV-C seront détaillés ici.

#### A. IBAD (Bus Address Register)

Ce registre permet d'attribuer une adresse au micro-contrôleur pour qu'il soit accessible en tant qu'esclave à partir d'un autre maître dans un environnement de type multi-maîtres. Mais, comme la maquette pédagogique est pilotée par un maître unique, en l'occurrence le micro-contrôleur HCS12, ce registre peut contenir une valeur quelconque.

#### B. IBFD (Bus Frequency Divider Register)

Ce registre de prévision de l'horloge permet de régler le taux de transfert sur le bus I2C. Le fonctionnement de ce registre est expliqué dans [2, § 3.3.2]. Un tableau contenant des choix différents de fréquences et un exemple illustratif y figurent également.

Ce registre a été réglé de manière à ce que le bus I2C soit cadencé à son débit maximum à savoir 100Kbit/sec en sélectionnant la valeur IBFD ← 0x47.

#### C. IBCR (Bus Control Register)

Comme son nom l'indique, ce registre permet de contrôler le fonctionnement du bus I2C.



Fig. 7. Registre de contrôle du bus I2C - IBCR

- IBEN (I-bus Enable) permet d'activer et de désactiver le module I2C du HCS12.

- MS/SL (Master/Slave mode select bit) permet de générer une condition de départ lorsque son état passe de

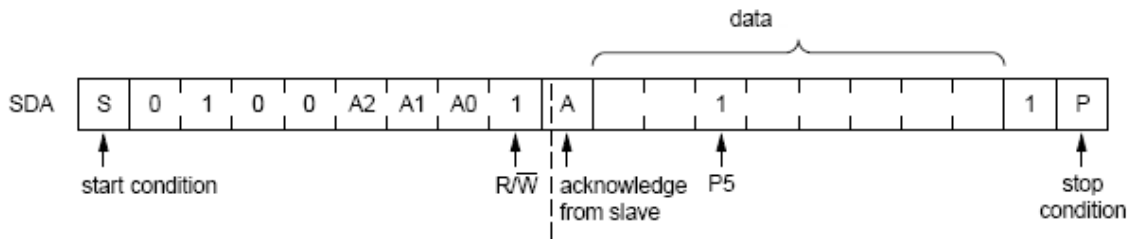


Fig. 6. Exemple d'une trame I2C complète

0 à 1 et une condition d'arrêt lorsque son état passe de 1 à 0.

- Tx/ $\overline{\text{Rx}}$  (Transmit/Receive mode select bit) permet de sélectionner le sens du transfert de données (lecture ou écriture).
- TXAK (Transmit Acknowledge enable) permet de configurer l'acquittement d'une trame par le HCS12.

#### D. IBSR (Bus Status Register)

Ce registre d'état permet, à travers ces différents bits, de connaître à tout moment l'état de la communication sur le bus I2C.

	7	6	5	4	3	2	1	0
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W								
RESET:	1	0	0	0	0	0	0	0

■ = Unimplemented or Reserved

Fig. 8. Registre d'état du bus I2C - IBSR

- TCF (Data transferring bit). Ce bit est à 0 pendant le transfert d'un octet et passe à 1 après. Cependant, ce bit ne permet pas à lui seul de savoir si un transfert de données est terminé ou pas. A cet effet, il est impératif de tester aussi le bit IBIF, même si le transfert ne se fait pas sous interruption.
- IBB (Bus Busy bit). Permet d'indiquer si le bus est occupé ou pas.
- IBIF (I-Bus Interrupt). Outre le rôle de ce bit dans l'indication et l'acquittement des interruptions, il joue également un rôle important dans l'indication de la fin de transfert de données.
- RXAK (Received Acknowledge). Ce bit indique si la trame a été acquittée par le récepteur. Il reflète donc l'état du bit 9 de la ligne SDA après le transfert d'un octet.

#### E. IBDR (Data I/O Register)

Si IBCR\_Tx\_Rx est à 1, une écriture dans ce registre enclenche un transfert de données sur le bus. Si IBCR\_Tx\_Rx est à 0, une lecture de ce registre enclenche un nouveau transfert de données, sauf si la condition d'arrêt a déjà été générée.

## IV. MAQUETTE PÉDAGOGIQUE

La maquette est constituée de deux parties. Un kit de développement (*starter kit*) à base du microcontrôleur HCS12 achetée dans le commerce et une carte d'extension développée localement contenant deux composants esclaves I2C.

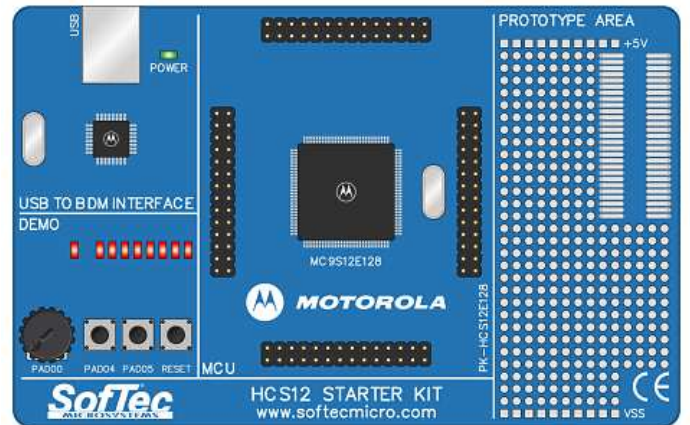


Fig. 9. Starter kit MC9S12E128 de Softec

#### A. Starter Kit MC9S12E128 de Softec

Commercialisé par la société Softec pour la somme de 100\$ (environ 75€), ce *starter kit* (fig.9) est livré avec le logiciel de développement Metrowerks de Motorola permettant de programmer le microcontrôleur HC12. Plusieurs périphériques du HCS12 sont ainsi programmables sans avoir à développer des cartes d'extension, comme les ports d'entrée/sortie, les convertisseurs analogiques numériques, numériques analogiques, etc. Le *starter kit* dispose également de plusieurs pins d'extension permettant de relier les différents modules du HC12 à des périphériques externes. Un circuit de développement de prototype est disponible à cet effet. Ce kit se connecte sur le port USB et ne nécessite aucune alimentation externe.

#### B. Développement de la carte d'extension I2C

La carte d'extension I2C est constituée principalement de deux composants : le PCF8574A permettant de convertir les trames I2C vers un port d'Entrée/Sortie parallèle [3] et le PCF8591, convertisseur analogique numérique, numérique analogique [4]. La simplicité du schéma électrique de la carte d'extension I2C de la fig.10 mérite d'être soulignée. Conformément aux spécificités du bus I2C deux résistances de rappel relient  $V_{cc}$  aux lignes SDA et SCL. L'adresse globale du composant PCF8574A, constituée de l'adresse constructeur 0111 et de l'adresse locale 010 correspondant au câblage des lignes  $A_2A_1A_0$ , est 0x74 en lecture et 0x75 en écriture. L'adresse globale du composant PCF8591, constituée de l'adresse constructeur 1001 et de l'adresse locale 000 est 0x90 en lecture et 0x91 en écriture.



Fig. 11. Carte d'extension I2C

Le PCF8591 dispose de quatre canaux de conversion analogique numérique Ain0 à Ain3 en entrée et d'un canal de conversion numérique analogique en sortie. Seuls les canaux Ain0 et Ain1 sont reliés à un potentiomètre conformément au schéma de la fig.10. La sortie analogique est quant à elle reliée à une LED dont la luminosité peut être commandée numériquement.

Il est important de signaler que le PCF8591 dispose d'un registre de contrôle interne (voir [4, fig.5]) permettant de sélectionner le canal de conversion et de configurer le mode de mesure des tensions : soit par rapport à la masse soit en différentiel.

### C. Développement logiciel

Dans une première approche, le module I2C du HCS12 peut être programmé en mode scrutation (polling). Une fois la programmation du module I2C maîtrisée, la deuxième étape consiste à programmer le module I2C sous interruption.

Seule la programmation du mode scrutation est détaillée dans cette communication. Le mode interruption est quant à lui traité dans la note d'application [1] ; les utilisateurs ont aussi la possibilité de télécharger les programmes gérant le bus I2C sous interruption. Cependant, l'utilisation de ces programmes devient rapidement un exercice de manipulation de pointeurs, tous les aspects afférents à la programmation du module I2C étant masqués.

Cinq fonctions, dont les algorithmes sont décrits ci-dessous facilitant la programmation du module I2C du HCS12, ont été développées. Le programme principal, se contente de faire des appels successifs à ces fonctions comme le montre l'algorithme 6.

#### C.1 Fonction d'initialisation du module I2C

Elle commence par configurer le registre de prédivision de l'horloge interne du microcontrôleur, puis active le module I2C conformément à l'algorithme 1.

---

**Algorithme 1** – Fonction InitI2C, entrées : néant, sortie : néant

---

```
IBFD ← 0x47
IBCR_IBEN ← 1
```

---

#### C.2 Fonction générant la condition de départ

Le principe de fonctionnement de cette fonction est de faire passer le bit MS\_SL du registre de contrôle IBCR de 0 (état originel supposé) à 1, ce qui permet de générer la condition de départ qui finit de se propager sur le bus lorsque ce dernier passe à l'état occupé.

---

**Algorithme 2** – Fonction Start, entrées : néant, sortie : néant

---

```
IBCR_MS_SL ← 1
tant que IBSR_IBB = 0 faire
    // attendre la propagation de la condition de départ sur le bus
fin tant que
```

---

#### C.3 Fonction générant la condition d'arrêt

Le principe de cette fonction est très similaire à la précédente. Une transition de 1 (état originel supposé) à 0 du bit MS\_SL du registre de contrôle IBCR, permet de générer la condition d'arrêt qui finit de se propager sur le bus lorsque ce dernier passe à l'état libre.

---

**Algorithme 3** – Fonction Stop, entrées : néant, sortie : néant

---

```
IBCR_MS_SL ← 0
tant que IBSR_IBB = 1 faire
    // attendre la propagation de la condition d'arrêt sur le bus
fin tant que
```

---

#### C.4 Fonction d'écriture

Pour générer une trame d'un octet sur le bus I2C, il suffit d'écrire cet octet dans le registre de données IBDR, le microcontrôleur se chargeant de sa sérialisation sur le bus. Le test de fin de transmission se fait simultanément sur les bits TCF et IBIF conformément à l'algorithme suivant :

---

**Algorithme 4** – Fonction Écriture, entrées : Data (octet), sortie : booléen

---

```
IBCR_TX_RX ← 1
IBDR ← Data
tant que IBSR_TCF = 0 ou IBSR_IBIF=0 faire
    // attendre la fin de la transmission
fin tant que
IBSR_IBIF ← 1 //RAZ d'IBIF
retour IBSR_RXAK
```

---

Il est clairement stipulé dans [2, p.36] qu'il ne suffit pas de tester le bit TCF pour savoir si une transmission est terminée ou pas. Nous avons ainsi constaté que ce bit passe à 1 avant la fin de la transmission. C'est le drapeau IBIF qui permet d'indiquer que la transmission s'est achevée. La nécessaire remise à zéro du drapeau IBSR\_IBIF se fait en écrivant 1 par dessus, comme pour tous les microcontrôleurs de la famille Motorola.

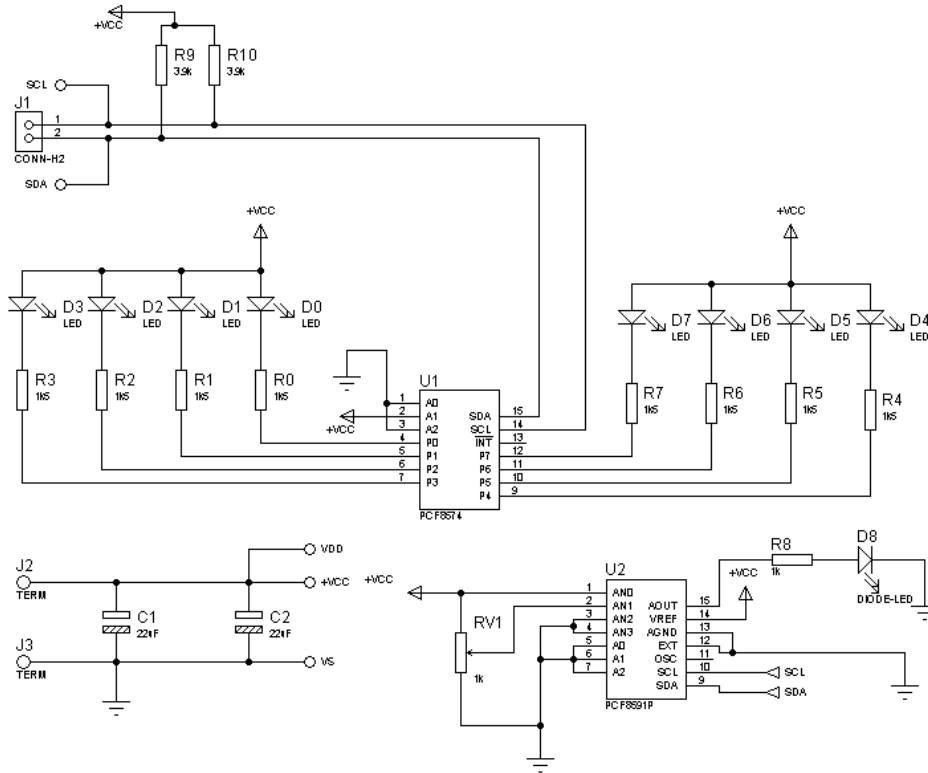


Fig. 10. Carte d'extension I2C

### C.5 Fonction de lecture

La fonction de lecture d'un octet est la plus délicate à mettre œuvre. Elle nécessite une compréhension approfondie de la manière dont le HCS12 gère la réception. Le paramètre `type` a été introduit dans l'algorithme 5, afin de pouvoir gérer les quatre cas possibles.

Tout d'abord, c'est le microcontrôleur qui pilote la transmission à partir des escalves, car c'est lui qui génère le signal d'horloge. C'est aussi sa responsabilité d'acquitter ou pas les trames reçues.

Une transmission est enclenchée lors de la lecture du registre de données (IBDR), d'où la nécessité d'effectuer une lecture factice afin que la première trame soit transmise (ce qui correspond au cas `type = 1`). Ensuite, le microcontrôleur peut recevoir et acquitter autant de trames de données que souhaitées (`type = 2`). Une subtilité supplémentaire vient s'ajouter : la dernière trame ne doit pas être acquittée afin que l'escalve interrompe la transmission des données ; or, la dernière trame est envoyée lors de l'avant dernière lecture, d'où la nécessité de ne pas acquitter l'avant dernière lecture (`type = 3`). Enfin, il faut générer la condition d'arrêt avant de lire le dernier octet reçu, afin d'empêcher une nouvelle transmission (`type = 4`).

### C.6 Fonction principale

La fonction principale se contente d'appeler successivement les fonctions développées. Après l'initialisation du module I2C, le programme principal (algorithme 6) rentre dans une boucle infinie qui effectue une conversion analogique numérique, récupère la valeur convertie, puis la renvoie vers le convertisseur numérique analogique et vers le port parallèle du PCF8574A.

---

### Algorithme 5 – Fonction Lecture, entrées : type (octet), sortie : octet

---

```

IBCR_TX_RX ← 0
si type = 1 //Lecture factice    alors
    IBCR_TXAK ← 0
sinon si type = 2 alors
    tant que (IBSR_TCF = 0 ou IBSR_IBIF=0) faire
        //attendre la fin de la transmission
    fin tant que
    IBCR_TXAK ← 0
sinon si type = 3 alors
    tant que (IBSR_TCF = 0 ou IBSR_IBIF=0) faire
        //attendre la fin de la transmission
    fin tant que
    IBCR_TXAK ← 1 //NACK de la trame reçue
sinon si type = 4 alors
    tant que (IBSR_TCF = 0 ou IBSR_IBIF=0) faire
        //attendre la fin de la transmission
    fin tant que
    stop()
fin si
IBSR_IBIF ← 1
retour IBDR //Lecture du registre de réception

```

---

Afin d'alléger l'écriture de l'algorithme, les tests d'acquitterment ont été volontairement supprimés.

### D. Analyse des trames

A tout moment, il est possible de visualiser et de décortiquer les trames I2C en observant les lignes SDA et SCL. Des exemples de trames acquises sont reportés sur le

**Algorithme 6** – Fonction Main, entrées : néant, sortie : néant

```

Variable : lu(octet), Ack(booléen)
InitI2C()
start() //Trame de config. du mot de contrôle du PCF8591
Ack ← ecriture(0x90)
Ack ← ecriture(0x40) //Ecriture du mot de contrôle du
PCF8591
stop()
tant que vrai faire
  start() //Trame de conv. A-N
  Ack ← ecriture(91) //Ecriture de l'@ du PCF8591
  lu ← lecture(1) //Lecture factice
  lu ← lecture(2) //Lecture normale
  lu ← lecture(3) //Avant dernière lecture
  lu ← lecture(4) //Dernière lecture; la condition d'arrêt est
générée dans lecture(4) afin d'empêcher la transmission d'une
nouvelle trame
  start() //Trame de conv. N-A
  Ack ← ecriture(0x90) //Ecriture de l'@ du PCF8591
  Ack ← ecriture(0x40) //Ecriture du mot de contrôle
  Ack ← ecriture(lu) //Ecriture de la donnée lue
précédemment
  stop()
  start() //Trame d'envoi de la valeur numérique lue sur le
PCF8574
  Ack ← ecriture(0x74)
  Ack ← ecriture(lu)
  stop()
fin tant que

```

fig.12.

### E. Documentation

La documentation nécessaire à la programmation du module I2C du HCS12 est [1], [2].

La documentation nécessaire au pilotage des composants I2C PCF8574A et PCF8591 est [3], [4].

La documentation nécessaire pour le fonctionnement du *starter Kit* MC9S12E128 est [5].

## V. CONCLUSION

Les solutions matérielles et logicielles présentées dans cette communication peuvent faire l'objet d'un projet d'une quarantaine d'heures pour la réalisation de cette maquette. Une fois tout en place, des travaux pratiques de 3 à 12h peuvent être mis en place, selon le degré d'approfondissement souhaité sur le bus I2C. Sur une séance de 3h, les étudiants peuvent écrire le programme principal et analyser les trames I2C. Sur une durée plus longue, ils peuvent écrire les différentes fonctions en commençant par les plus simples (Start, Stop, Ecriture). Il est aussi possible de demander aux étudiants de câbler les composants I2C.

## RÉFÉRENCES

- [1] Motorola. Note d'application, AN2318/d, rev. 0,8/2002, using the I2C bus with HCS12 microcontrollers. 2002.
- [2] Motorola. HCS12 Inter-Integrated circuit (IIC), Block Guide V02.08. Juin 2004.
- [3] Philips Semiconductors. Datasheet du PCF 8574, remote 8-bit I/O expander for, I2C-bus. 1997.

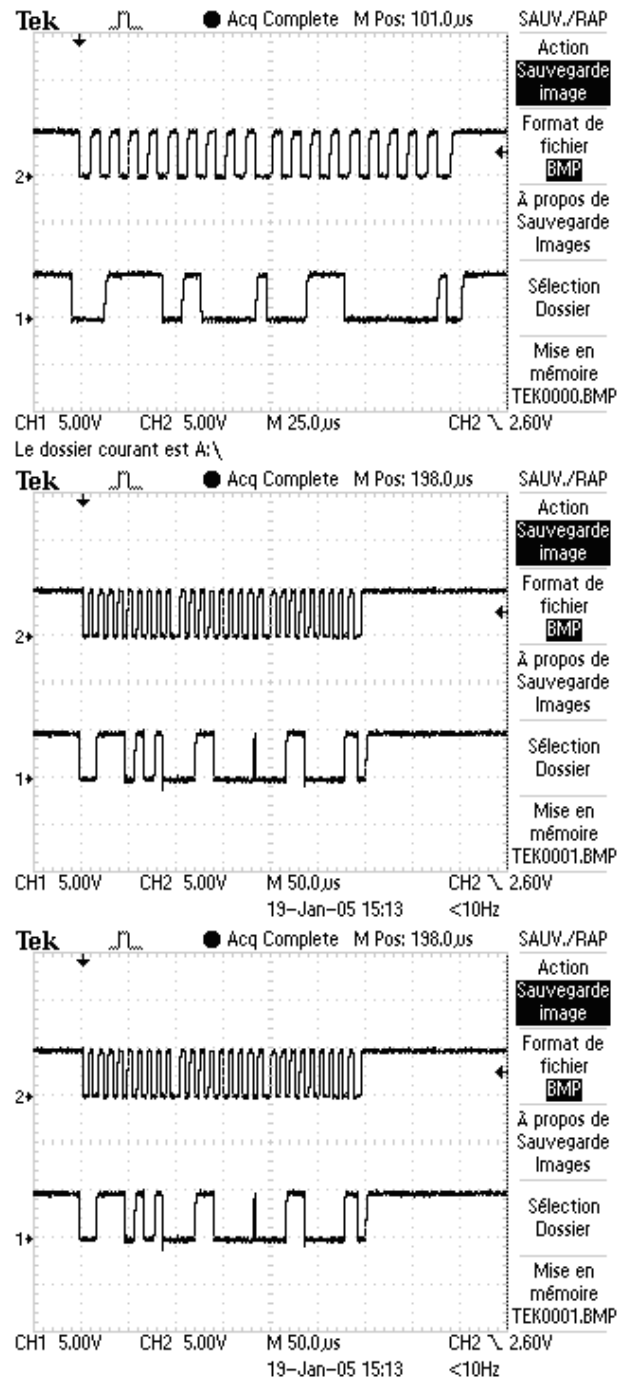


Fig. 12. Analyse de trames I2C

- [4] Philips Semiconductors. Datasheet du PCF 8591, 8-bit A/D and D/A converter. 2003.
- [5] Softec. PK-HCS12E128 user's manual. 2002.