Mise en place d'une communication entre un 68332 et un PIC pour dialoguer sur un bus I2C

Florent de LAMOTTE

12 septembre 2002

Ce document présente la mise en place d'une communication entre un 68332 et un PIC16F877 à travers un bus parallèle. Nous avons dù mettre en place cette solution dans le cadre de la Coupe de France de Robotique 2002 pour laquelle nous avions opté pour une solution multicartes communicant sur un bus I2C. Pour uniformiser les interfaces avec le bus, nous avons décidé de ne placer sur le bus que des PIC877. Le processeur principal du robot étant un 68332 qui devait donc dialoguer avec un PIC pour envoyer des données sur le bus.

Il peut être interessant, avant de lire ce document, de lire le polycopié de M.Perdriau (polycopié ESEO) qui est une très bonne synthèse de la datasheet à défaut, vous pouvez lire la datasheet du 68332.

1 Les bêtes

1.1 Le 68332

Le 68332 (bien connu des étudiants de l'ESEO) est un microcontrôleur Motorola de la famille CPU32. Il possède un coeur CPU32 32 bits descendant du très célèbre 68000 dont le jeu d'instructions est très proche de celui du 68020 (moins quelques instructions). Autour de celui-ci gravitent plusieurs composants citons le SIM (System Integration Module) qui gère l'interface bus, un watchdog, un timer, des Chip-Selects et de nombreux ports, le QSM (Queued Serial Module) permettant notemment de gérer la liaison série et le TPU (Time Processor Unit) une unité de calculs travaillant sur les signaux temporels.

1.2 Le PIC 16F877

Le PIC 16F877 est à ce jour le plus gros microcontrôleur de la série 8/12 bits de Microchip. Il possède entre autre une interface série RS232 une interface I2C, un PSP (Parallel Slave Port) port 8 bits permettant au PIC de s'interfacer en tant que périphérique sur le bus de données d'un autre microcontrôleur où d'un microprocesseur.

Grâce au PSP, on va donc être capables de s'interfacer sur le bus de données du 68332 et agir en tant qu'esclave.

2 L'interfaçage

2.1 Les signaux utilisés

Du côté du 68332, en plus des broches du bus de donnée et du signal R/_W, nous utiliserons un chipselect (_CS2) et une interruption (PF5), l'interruption correspondante sera levé par le PIC à chaque fois qu'une donnée pourra être lue par le 68332.

Du côté du PIC, nous devrons réserver pour la communication les signaux :

– _RD et _WR qui nous permettrons de connaître le sens de la communication (lecture/ecriture)

– _CS qui permet de valider une lecture ou une écriture sur le PIC
– PSP[0-7] qui seront interfacées sur le bus de données du 68332
– RB0 qui connectée à une broche d'interruption du 68332 nous permettra de lui faire savoir qu'on a quelque chose à lui dire.

2.2 La conversion des R/_W

Le 68332 utilisant un bus à la norme Motorola (un signal R/_W pour spécifier le sens de la communication et un signal _DS pour dire que le bus est prêt) et le PIC utilisant la norme Intel (On abaise _RD ou _WR pour dire que la donnée sur le bus est prête pour être lue ou écrite) il nous faut utiliser un petit circuit constitué de quatres portes NAND pour transformer R/_W + _CS (sur le 68332 on peut configurer les chip-selects pour qu'ils soient activés sur _AS ou _DS) en _CS + _RD + _WR.

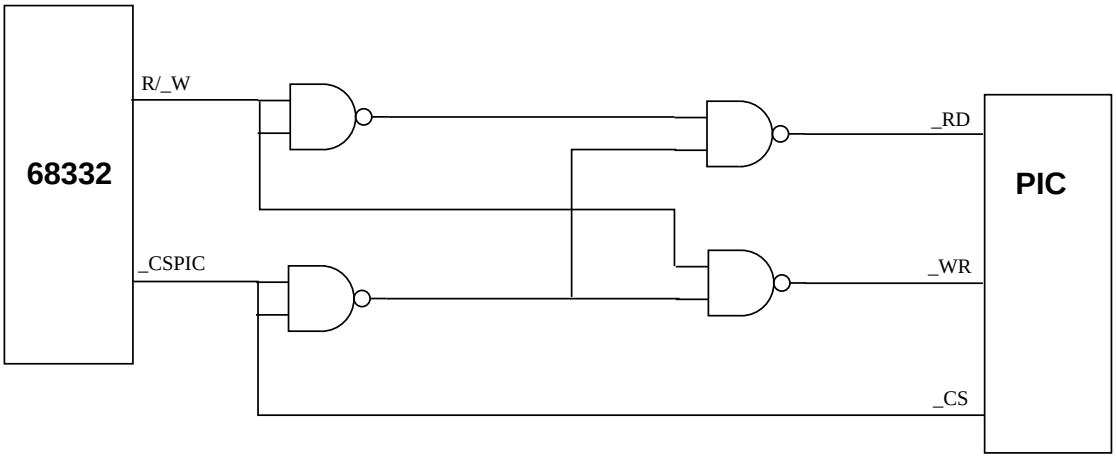

FIG. 1 – Conversion des signaux de bus

3 Le fonctionnement du PSP

4 La configuration du 68332

4.1 Le chip-select du PIC

Nous avions décidé que le chip-select correspondant au PIC serait _CS2 et que l'adresse de base du PIC serait 0x100000.

Nous avons donc placé la valeur 0x1000 (adresse de base 0x100000 bloc de taille minimale (2k)) dans CSBAR2 et la valeur 0x7C70 (R/W, 3 wait states, mode superviseur) dans CSOR2.

Il faut configurer les chip-select en mode 8 bits pour que les données soit bien placées sur la bonne partie du bus (les bits de poids fort).

Notre configuration du chip-select est donc la suivante : CSPAR0 bits 6 et 7 -> '10'. (CSPAR0 etant le registre permettant la configuration du port C)

4.2 L'interruption

La configuration des pins du PortF en tant qu'interruption ou Port peut se fait de manière globale dès le démarrage à l'aide de la broche D9 (Un pulldown sur D9 configure le port en tant que port d'entrée sortie). Il

est possible ensuite de configurer indépendemment les pins du port à l'aide du registre PFPAR.

Il est obligatoire, si certaines pattes du port F n'ont pas d'état déterminé au démarrage de configurer le port F pour agir comme un port d'entrée/sortie. En effet, si une patte du port F est flottante, elle sera en mesure de déclencher une interruption non désirée.

Dans notre cas le PortF était configuré pour agir comme un port d'entrée sorties dès démarrage. Nous validion donc l'interruption sur PF5 en passant le bit 5 du registre PFPAR a 1.

Il faut ensuite acquiter l'interruption. Cet acquitement se fait à l'aide d'un chip-select non utilisé : CS5. Il faut le configurer en tant que chip-select 8 bits on place donc les valeurs 1 et 0 dans les bits 12 et 13 de CSPAR0.

0n mets CSBAR5 à 0xFFF8 (adresse avec des 1 uniquement)

Et on configure CSOR5 pour avoir :

- BYTE=0 (mode 8 bits)
- R/_W=1 (lecture seulement)
- SPACE=0 (CPU)
- IPL=5 (on acquite l'interruption de niveau 5 mais on aurait pû mettre 0, cela aurait acquité toutes les interruptions)
- _AVEC=1 on veut untiliser une interruption autovectorisée

Nous avons maintenant une interruption vectorisée de niveau 5, nous plaçons donc dans l'auto-vector de niveau 5 l'adresse de la fonction qui va traiter notre interruption.

Il ne faut pas oublier d'acquiter l'interruption. Dans notre cas, une lecture sur le PIC suffira pour acquiter l'interruption.

5 Les routines de bas niveau

Le code gérant la communication entre le 68332 et le pic se trouve dans le fichier pic_i2c.c, les déclarations concernant l'i2c se trouvant dans i2c.h.

Les routines de bas niveau, gérant l'i2c sont utilissées principalement par les routines de haut niveau. On trouve dans ces routines :

- L'interruption gérant le pic, nommée I2C_int. Cette fonction va ajouter l'octet envoyé par le pic dans le buffer circulaire. Une fois la donnée placée dans le buffer l'index de debut de buffer est incrémenté.
- Une fonction de lecture dans le buffer : read_PIC. Celle-ci va attendre qu'une donnée soit présente dans le buffer de récéption. Une fois qu'une donnée est arrivée, celle ci est récupérée et l'index de fin de buffer est incrémenté.
- La fonction send_PIC, qui envoie une donnée au PIC
- La fonction empty_i2c_buffer, chargée de vider le buffer circulaire en plaçant les deux index à la même valeur.
- La fonction I2C_reset qui reboote le PIC

Ces routines s'occupent donc principalement de gérer les temporisations à respecter entre chaque communication avec le PIC, ainsi que du buffer circulaire i2c_buffer.

Nous avons aussi écrit trois macros s'occupant de gérer l'accès vers le pic. En effet, il faut être très vigilant et ne pas accéder en même temps au pic dans la boucle principale et dans l'interruption. De plus il faut éviter d'attendre le pic dans l'interruption car dans ce cas on l'attendrait une éternité.

- *get_pic* attends que le pic soit libre
- *test_pic* idem, mais si le pic n'est pas libre continue au release_pic suivant
- *release_pic* réautorise l'accès au pic

Voici l'implémentation de ces macros définies dans *i2c.h* :

```
volatile byte pic_used;
```

```
#define test_pic if(!pic_used && !pic_block_it) {\
                pic_used = 1;

#define get_pic while(pic_used);{\
                pic_used = 1;\
                pic_block_it = 1;

#define release_pic pic_used = 0;}
```

6 Les routines de haut niveau

Les routines de haut niveau utilisent les routines de bas niveau pour dialoguer avec le PIC et sont chargées de commander le programme du PIC.

Ces routines sont au nombre de trois :

– I2C_send est chargé d'envoyer des données à d'autres cartes par l'intermédiaire du PIC.
– I2C_recv reçoit des données en provenance d'autres cartes
– I2C_set_type_r positionne le type de retour d'une carte

6.1 I2C_send

Cette fonction est chargée d'envoyer des données à d'autres cartes par l'intermédiaire du PIC.

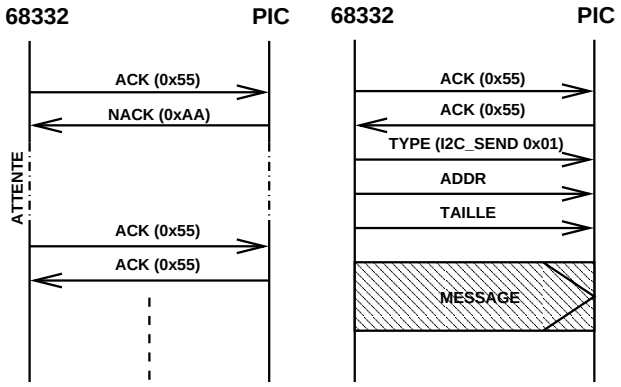

FIG. 2 – Envoi de données vers le PIC

La communication avec le PIC est détaillée à la figure 2.

Dans cette communication, le 68332 est toujours maître. Il commence par envoyer un signal d'acquitement (ACK 0x55), et attends du PIC le même signal. Si le PIC envoie un signal de non acquitement (NACK 0xAA) cela signifie qu'il est occupé et qu'il ne peut pas satisfaire notre demande. Il faut donc attendre puis réessayer.

Une fois la poignée de main terminée, il faut envoyer les données au PIC pour qu'il puisse les passer par le bus I2C. On commence par envoyer le type de commande que l'on veut faire exécuter par le PIC. Ici nous voulons envoyer une donnée sur le bus I2C, nous utiliserons donc la commande I2C_SEND soit 0x01.

Il faut ensuite indiquer au PIC à qui nous allons envoyer ces données, par l'intermédiaire du champ ADDRESSE qui est un octet correspondant à l'addresse de la carte à laquelle on veut envoyer des données.

Le pic a besoin de connaître le nombre d'octets qu'il va recevoir du 332 et envoyer sur le bus, ceci se fait à l'aide de l'octet TAILLE.

Pour finir on envoie le message de la taille spécifiée au PIC. Une fois le message envoyé, le PIC se charge de son transfert sur le bus.

6.2 I2C_recv

La récéption de données s'éffectue quasiment de la même manière que l'emmission de donnée. La carte principale étant maître sur le bus I2C, c'est elle qui doit initier les communications avec les autres cartes même quand ce sont les autres cartes qui doivent envoyer des données à la carte principale.

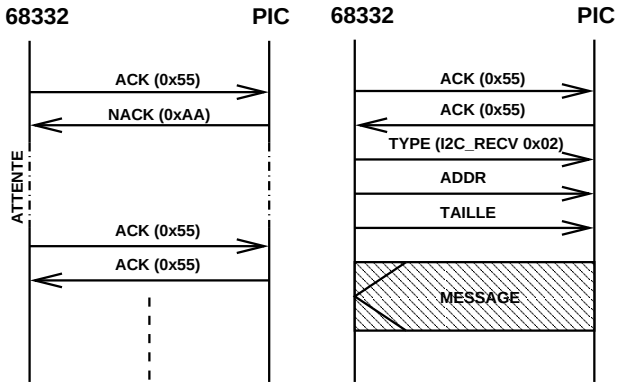

FIG. 3 – Réception de données depuis le PIC

Comme le montre la figure 3, la réception de données en provenance du PIC est analogue à l'emmission de données vers le PIC.

Il faut donc encore une fois effectuer la poignée de main, envoyer le type de message qui sera ici une réception de données, préciser la taille et se préparer à recevoir des données en provenance du PIC.

Cela peut paraître bizarre d'avoir à préciser la taille des données que l'on va recevoir. Il faut bien se souvenir que le 68332 est maître et donc demande une information aux cartes esclaves. Chaque type d'information possède une taille précise comme par exemple les données du type status qui sont codées sur un seul octet. Le type de données que l'on souhaite recevoir de la carte esclave est précisé à l'aide de la commande SET_TYPE_R explicitée juste après.

6.3 I2C_set_type_r

La commande I2C_set_type_r permet de demander à une carte esclave de se préparer à envoyer une donnée d'un type précis.

Il s'agit tout bêtement d'une commande d'envoi vers la carte esclave intéressée d'une commande SET_TYPE_R suivie d'un code correspondant à l'information que l'on souhaite recevoir (une position, un status, ...).

Jusqu'à ce que cette valeur soit changé à l'aide d'une autre commande SET_TYPE_R, chaque demande de récéption en provenance de cette carte retournera l'information demandée.