Projet de semestre :
Carte ARM XScale

Julien Pilet & Stéphane Magnenat
Informatique huitième semestre

Juin 2002
$Revision : 1.6 $

Laboratoire d'Architecture des Processeurs
Faculté d'Informatique et Communication
Professeur : Paolo Ienne
Responsable : René Beuchat

Résumé

The goal of this project is to build an Intel XScale based motherboard for the Cyclope robot developed at LAMI (now LAP). The Intel PXA250 has been chosen for its performances and low power consumption.

To decrease the complexity, the system has been split into 3 components : an alimentation board, a processor board and one or more extension board. The alimentation board has been designed in cooperation with Cedric Gaudin who has also designed a card for the Cyclope. For the purpose of compatibility, two buses have been defined. The A-BUS connects the processor board to the alimentation ; and the Milli-BUS connects the processor board to the extensions cards.

For software development a completely free solution has been chosen. The compilation toolchain is based on gcc 3 and gdb 5.2. A home made JTAG interface based on an USB microcontroller board allows us to upload code into the processor.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 But du projet

Ce projet consiste à développer un système embarqué autonome pour contrôler un robot mobile Cyclope avec un processeur de type ARM. Une carte a été réalisée en portant une attention particulière à la consommation.

La conception de cette carte doit être suffisamment générique pour d'autres usages, tel que de servir de support de travaux pratiques. Elle doit en conséquence pouvoir fonctionner de manière autonome, sans le robot Cyclope.

L'utilité de cette carte dépendra en grande partie de sa puissance de calcul et de ses périphériques. C'est pourquoi l'accent a été mis sur l'évolutivité des extensions, une grande mémoire vive et un emplacement CompactFlash qui permet de fournir une mémoire de masse importante. Ces atouts, alliés à la puissance fournie par le processeur XScale d'Intel, à la petite taille de la carte et à sa consommation modérée, promettent un résultat intéressant. Les outils de développement GNU garantissent une solution de développement gratuite, fonctionnelle et adaptable à nos besoins.

La figure 1.1 montre différentes utilisations possibles de la carte PXA250.

1.2 Autres cartes disponibles pour le Cyclope

Deux carte existent déjà pour le robot cyclope.

La première, basée sur un 68HC12 de Motorola, possède 32K de RAM et est bien adaptée au développement en assembleur. Il est aussi possible de faire du C, mais le processeur 8 bits n'est pas très adapté aux compilateurs modernes.

La seconde est une carte basée sur le 68336 de Motorola. Elle possède 512K de RAM. Le processeur 32 bits permet de travailler confortablement, que ce soit en assembleur, C ou C++.

Notre carte, nettement plus puissante que ces dernières, permettra d'aller encore plus loin. Il sera possible de faire du traitement d'image en temps réel (et pas seulement sur une seule ligne de 102 pixels), d'embarquer des algorithmes complexes qui demandent beaucoup de mémoire et de temps de calcul ; par exemple des réseaux de neurones ou des algorithmes génétiques.

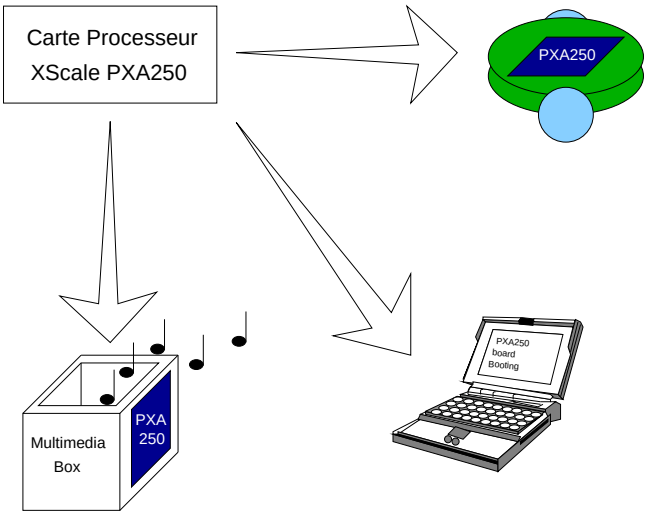

FIG. 1.1 – Utilisations possibles de la carte XScale.

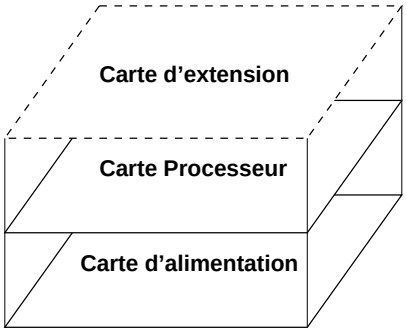

FIG. 1.2 – Organisations des trois différentes parties du système.

Pour les inconditionnels de la perte d'énergie inutile, il est parfaitement envisageable de faire tourner une machine virtuelle java, et pour les plus téméraires, du C#.

1.3 Description générale du système

A cause de notre manque d'expérience en matière de conception de PCB et de la nouveauté du processeur, nous avons voulu limiter au maximum la portée des erreurs potentielles. Ainsi, nous avons choisi un système à 3 cartes. Une carte d'alimentation, une carte processeur et éventuellement une ou plusieurs cartes d'extension. Le processeur est le XScale [1] de Intel, descendant du StrongArm, et implémentant le jeu d'instruction ARM5.

Cédric Gaudin [7] faisant une carte FPGA pour le robot cyclope, nous avons décidé d'avoir une carte d'alimentation compatible. Le bus connectant la carte

processeur à celle d'alimentation est nommé le A-BUS (description en annexe). De plus, désirant être le plus compatible possible, nous avons aussi défini un connecteur d'extension commun, nommé le Milli-BUS (description en annexe). Ainsi, on peut concevoir des cartes d'extension fonctionnant avec les deux systèmes.

En règle générale, les connecteurs mâles sont sur la face supérieure de la carte (les connecteurs d'extensions). Les connecteurs femelles sont contre le bas (alimentation).

Chapitre 2

Conception de la carte d'alimentation

La carte d'alimentation est le lien entre la carte processeur et le robot cyclope. Tout en s'occupant de l'alimentation, elle fournit aussi la logique nécessaire pour connecter le A-BUS au cyclope.

On peut voir sur la figure 2.1 le schéma bloc de cette carte.

2.1 Alimentation

La partie alimentation de cette version ne contient que des points de tests destinés à recevoir du courant d'alimentations externes. Une prochaine version devrait contenir les convertisseurs nécessaires à l'alimentation depuis des piles ou des batteries.

Les tensions fournies sont : 5V, 3.3V, variable 0.85V - 1.3V, et 1.8V. Une alimentation de secours 3.3V peut être séparée ou identique à l'alimentation 3.3V principale, le choix se faisant à l'aide d'un cavalier.

2.2 Driver moteur

Deux drivers SI9986CY [8] attaquent les moteurs du Cyclope. Ils sont connectés aux signaux PWM de la carte processeur à travers une simple logique câblée car notre processeur n'a que deux canaux PWM.

2.3 Convertisseur A/D

Cette carte sert aussi d'acquisition pour les capteurs grâce à un convertisseur A/D 8 canaux de Maxim MAX1245BCAP [9]. Ce dernier est accédé par le bus SPI intégré au A-BUS. Les quatre premiers canaux sont utilisés pour les détecteurs de choc (avant droite, avant gauche, arrière droite, arrière gauche). Le cinquième permet de connaître le niveau des batteries, le sixième est relié aux capteurs de distance, le septième à caméra linéaire et le huitième est libre.

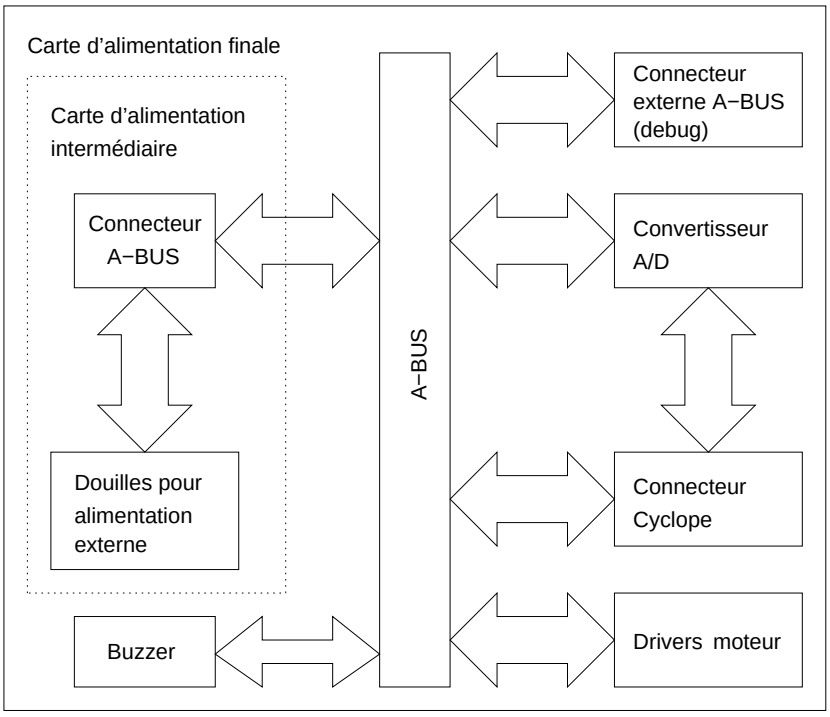

FIG. 2.1 – Schéma bloc de la carte d'alimentation.

Chapitre 3

Conception de la carte processeur

Cette carte, dont le schéma bloc est illustré par la figure 3.1, est la composante principale du système. Autonome, elle n'a besoin que d'alimentations pour fonctionner. Ce chapitre détaille les composants et connecteurs dont elle est munie, en expliquant les détails et contraintes des solutions choisies.

3.1 Processeur

Le processeur est un PXA250 [10] de chez Intel. C'est un circuit de 256 pattes en BGA, qui implémente un coeur XScale [1] à 200, 300 ou 400 MHz, un cache de 2x32KO ainsi qu'un nombre impressionnant de périphériques programmables. La figure 3.2 montre le schéma bloc du processeur.

3.2 Quartz

La documentation d'Intel spécifie qu'il suffit de brancher deux quartz de 32.768 kHz et de 3.6864 MHz sur les pins PXTAL et PEXTAL du PXA250. Néanmoins, nous avons préféré ajouter deux capacités et une résistance (facultatives) pour s'assurer du fonctionnement.

3.3 Mémoire FLASH

La mémoire Flash est une AS29LV800B-80TCB [11] de 1 MO sur 16 bits. Il n'y a pas de protection matérielle contre l'effacement.

Cette mémoire est branchée sur le chip select 0. Elle est visible du processeur à partir de l'adresse 0x00000000 qui est l'adresse de démarrage.

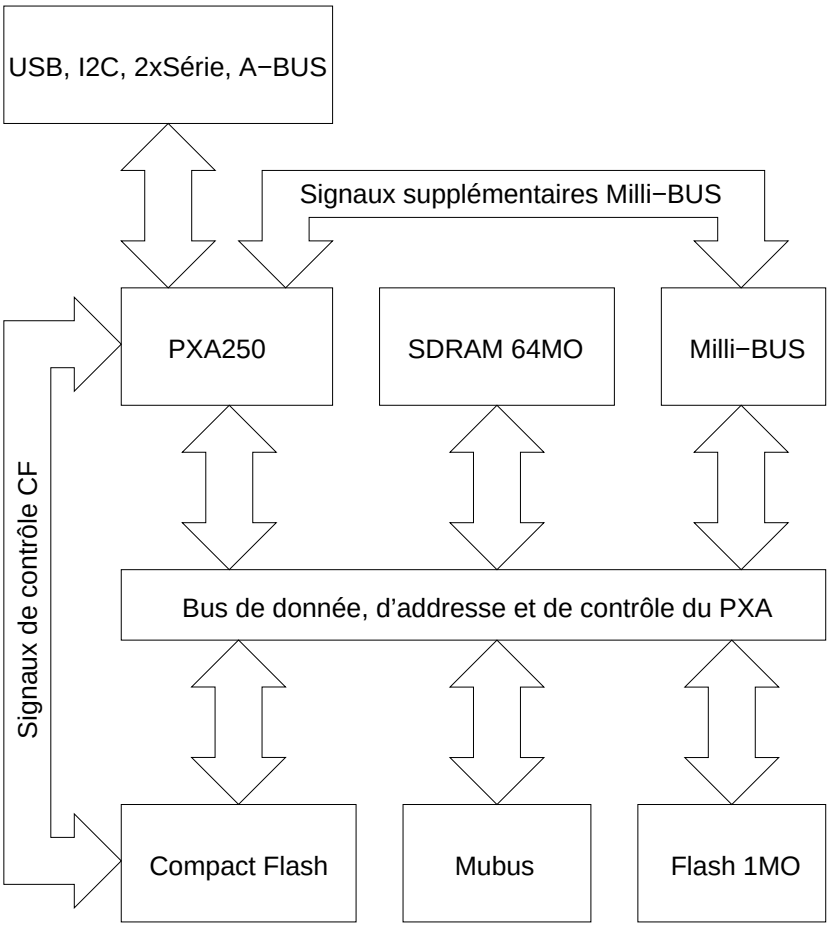

FIG. 3.1 – Schéma bloc de la carte processeur.

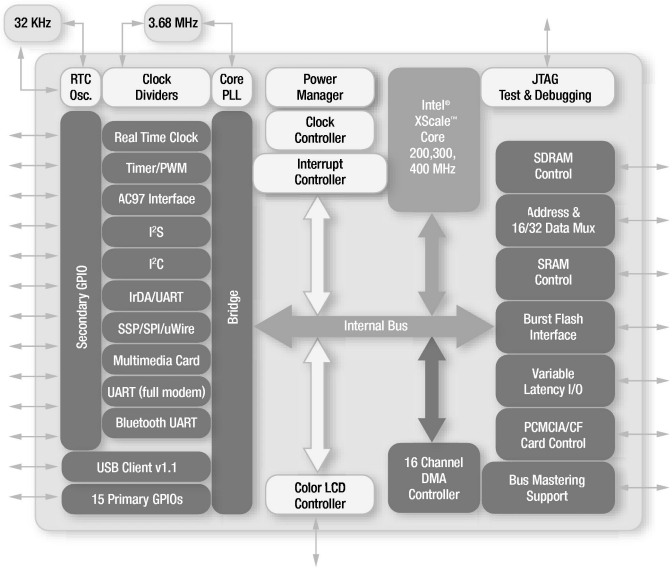

FIG. 3.2 – Schéma bloc PXA250 d'Intel, tiré de : *The Intel PXA250 Applications Processor - White Paper*.

3.4 Mémoires SDRAM

La mémoire SDRAM est composée de 4 chips V54C3128804VAT7 [12] de 128 MBits chacun. Ils sont branchés en parallèle de façon à former un bus mémoire de 32 bits de large permettant ainsi d'avoir 64 MO de RAM. Ces quatre chips occupent un des quatre emplacement SDRAM que gère le contrôleur de mémoire dynamique du PXA250.

3.5 Reset

L'utilisation d'un MAX809 [13] garanti que le signal RESET* s'active au minimum pendant 150 ms. Ce signal est partagé avec le JTAG qui le laisse en l'air lors de l'utilisation normale, ou le force à 0 pour effectuer un reset. Le signal RESET_OUT* sort du processeur et se désactive dès que le PXA250 est prêt à fonctionner.

3.6 Mubus

Le Mubus est connecté sur le chip select 5. Étant donné que le contrôleur ne gère que des périphériques 16 ou 32 bits et que le processeur travaille sur 32 bits, tous les accès au Mubus doivent se faire sur 32 bits. Seul les 8 bits de poids faibles sont écrits ou lus sur le Mubus. Il est accessible à partir de l'adresse 0x140000, jusqu'à 0x1400FC par pas de 4 octets.

3.7 CompactFlash

Un connecteur CompactFlash (CF) [14] est disposé sous la carte, permettant ainsi d'insérer une carte CF sans que cette dernière ne dépasse.

La spécification CF requiert d'accepter en entrée un voltage de 5V. Toutes les lignes en entrée sont passent donc par des Cybus 3384 pour assurer une tension maximale de 3.3V.

3.8 Connecteur d'alimentation

La carte processeur est alimentée à travers le A-BUS dont la spécification est donnée en annexe. Trois voltages sont nécessaires : 5V, 3.3V, et une alimentation variable entre 0.85V et 1.3V. Cette dernière pourrait être contrôlée par le processeur, via le bus I2C.

La fréquence du processeur est limitée par cette tension variable. Il convient de prendre des précautions particulières lors d'un changement d'alimentation, décrites dans deux documents d'Intel [4, 5].

Des signaux de contrôles généraux, spécifiques à l'alimentation et spécifiques au robot Cyclope sont également présents sur ce connecteur :

- I2C,
- 2 odomètres,
- SPI,
- 2 PWM,
- haut-parleur,
- 3 signaux de contrôle pour l'acquisition des capteurs de chocs, de distance et la caméra du cyclope (OPTSEN_CLK, OPTSEM_RST, OPTSEN_SI),
- signaux de contrôle d'alimentation : VDD_FAULT, BATT_FAULT, WAKEUP, PWREN et RESET.

Tous ces signaux sont détaillés dans l'annexe spécification A-BUS.

3.9 Connecteur d'extension

La carte contient le connecteur d'extension Milli-BUS dont la spécification est donnée en annexe. Celui-ci sort beaucoup de lignes de périphériques intégrés (LCD, AC97, etc.) qui peuvent être utilisées comme entrées/sorties générales (GPIO). Les bus d'adresse et de donnée, en conjonction avec les différents signaux de sélection de circuit, permettent d'ajouter de la mémoire ou autre périphérique à accès parallèle.

3.10 Connecteur USB

Le PXA250 a un contrôleur USB [15] esclave intégré. Pour l'utiliser, il faut mettre le signal USB_CONNECT branché sur le port GPIO 36 à 1. Cela permettra à l'hôte de voir un client connecté et de lancer la configuration. Pour savoir si un maître est branché, il faut lire le port GPIO 27 (USB_DETECT). Si on lit 1 il y a un maître présent.

3.11 Connections séries (RS232, I^{2}C)

La carte possède 2 ports séries asynchrones, un port I^{2}C et un port SSP. Adjoint d'un adaptateur de tension 0, 3.3V <-> -12, 12V, les 2 ports asynchrones sont compatibles RS232. Ils possèdent 2 connecteurs chacun, un MM6 et un SIL4 et sont directement connectés au processeur avec toutefois des pull-ups sur RX et TX. Le port I^{2}C est connecté sur un MM4 à travers un Cybus 3384 permettant ainsi de convertir les 5V de l'I^{2}C en 3.3V. Les pull-ups 5V nécessaire au bon fonctionnement de l'I^{2}C sont aussi présentes.

3.12 Connecteur JTAG

Le connecteur JTAG est un MM10. Il est compatible avec celui de la carte de Cédric Gaudin, permettant d'utiliser la même interface physique pour les deux projets.

Une interface JTAG USB basée sur le chip EZ-USB permet de charger du code dans le processeur (voir Système de démarrage).

Chapitre 4

Outils de développement

La chaîne d'outils de développement que nous avons choisie est celle de GNU. Ces outils, entièrement gratuits et libres, semblent performants. Néanmoins, il est difficile de juger de leur qualité sans plate-forme physique pour les tester.

4.1 Compiler le cross-compilateur

Nous avons compilé les outils suivants pour l'architecture XScale :

binutils 2.12 assembleur, manipulation de fichiers objets

gcc 3.0.4 cross-compilateur C et C++

gdb 5.2 débogueur comprenant un simulateur de noyau XScale

newlib 1.10.0 librairie C spécialement conçue pour des systèmes embarqués

linux 2.4.18 (avec patch *rmk5* et *rmk5-pxa1*) fichiers *.h* systèmes. Linux n'a pas été compilé, mais uniquement configuré pour le PXA250.

Le document *The GNU Toolchain for ARM targets HOWTO [18]* explique comment compiler ces outils. Voici le détail des opérations :

1. décompresser les archives *binutils*, *gcc*, *gdb* et *newlib* dans le même sous-répertoire (dans notre cas /home/pxa/src-all/) ;
2. installer les sources de Linux modifiée dans le répertoire /home/pxa/xscale-elf/xscale-elf/linux ;
3. depuis le répertoire de compilation (par exemple /home/pxa/xscale-elf/build/) lancer la configuration :

```
/home/pxa/src-all/configure --prefix=/home/pxa/xscale-elf \
--target=xscale-elf \
--with-cpu=xscale \
--with-newlib \
--with-headers=/home/pxa/xscale-elf/xscale-elf/linux/include \
--disable-threads \
--enable-languages=c,c++ \
--disable-shared \
--disable-nls
```

4. lancer la compilation avec : make
5. installer la suite de programmes dans le répertoire /home/pxa/xscale-elf/ avec la commande : make install.

Nous pouvons maintenant tester le fonctionnement du système en compilant un fichier C et simulant son exécution dans le simulateur interne à *gdb.* Tout semble fonctionner correctement.

4.2 Exemple d'utilisation du cross-compilateur et du cross-débogueur

Voici le fichier d'exemple test1.c :

```
int main(int argc, char *argv[])
{
int i;
int j;
i=0;
j=0;
while (i<20)
{
i++;
j-=i;
}
return 0;
}
```

Ces quelques commandes le compilent :

```
xscale-elf-gcc -g -c -o test1.o test1.c
xscale-elf-gcc -g -o test1 test1.o
```

On peut ensuite lancer la simulation :

```
> xscale-elf-gdb test1
GNU gdb 5.2
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=xscale-elf"...
(gdb) target sim
Connected to the simulator.
(gdb) load
Loading section .text, size 0x34d8 vma 0x8000
Loading section .rodata, size 0x38 vma 0xb4d8
Loading section .data, size 0x858 vma 0xb610
Loading section .eh_frame, size 0x4 vma 0xbe68
```

```
Loading section .ctors, size 0x8 vma 0xbe6c
Loading section .dtors, size 0x8 vma 0xbe74
Start address 0x8000
Transfer rate: 125920 bits in <1 sec.
(gdb) break main
Breakpoint 1 at 0x8100: file test1.c, line 5.
(gdb) run
Starting program: /home/pxa/xscale-elf/test/test1

Breakpoint 1, main (argc=1, argv=0x1ffffc) at test1.c:5
5               i=0;
(gdb) step
6               j=0;
(gdb)
7               while (i<20)
(gdb)
9                       i++;
(gdb)
10                      j-=i;
(gdb) info registers
r0             0x0      0
r1             0xbe6c   48748
r2             0x1      1
r3             0x1      1
r4             0xb624   46628
r5             0xb624   46628
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x200100 2097408
r11            0x1ffff8 2097144
r12            0xb87c   47228
sp             0x1fffdc 2097116
lr             0x8100   33024
pc             0x8128   33064
fps            0x0      0
cpsr           0x80000013       -2147483629
(gdb)
```

L'étape suivante consiste à utiliser *gdb* directement sur le processeur XScale, par l'intermédiaire du port JTAG. Pour ce faire, deux possibilités s'offrent à nous : modifier *gdb* pour qu'il supporte notre interface JTAG par USB (voire §5.7) ou acheter une solution existante. Par exemple, la solution proposée par Abatron [19] semble adaptée.

Chapitre 5

Système de démarrage

5.1 Problème du démarrage

Le processeur PXA250, dès que la ligne RESET se désactive, charge la première instruction à exécuter à l'adresse 0, qui correspond à la mémoire FLASH soudée sur la carte. Lors du premier démarrage, elle ne contiendra rien d'utilisable. Il faut donc un moyen externe pour influencer le démarrage.

Le système JTAG du PXA250 permet de résoudre ce problème en chargeant un logiciel dans la cache d'instruction avant que le processeur ne démarre. Le problème suivant consiste à piloter ce port JTAG. Après avoir cherché des solutions existantes et déploré leur prix, nous avons décidé de développer notre propre système décrit par la figure 5.1.

Ce système se compose d'une partie matérielle et d'une partie logicielle. Pour attaquer le port JTAG du PXA250, nous avons choisi une carte EzUSB développée au LAP. Le port parallèle du PC aurait été un autre choix possible, avec la création d'une petite carte intermédiaire adaptée. L'USB nous a séduit par sa portabilité, sa vitesse et parce que la carte était déjà fonctionnelle.

Le logiciel de contrôle sur PC se nomme *jtag-usb*. Nous verrons le principe de son fonctionnement (§ 5.4), puis son utilisation (§ 5.5) et finalement des détails à propos de son implantation (§ 5.6).

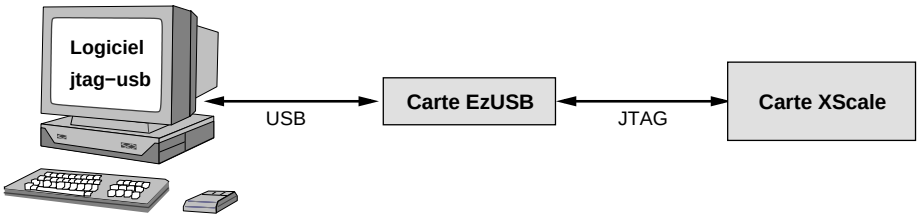

FIG. 5.1 – Connexion entre un PC et le PXA250. Le programme *jtag-usb* pilote le microcontrôleur *EZ-USB* connecté au port JTAG du processeur.

5.2 Interface JTAG basée sur le EZ-USB

Nous avons développé notre propre interface JTAG. Celle-ci se base sur le chip EZ-USB de Cypress [16] qui est un microcontrôleur USB basé sur le 8051. Il contient une implémentation du protocole USB esclave en matériel et est assez facile à utiliser. Sur ce circuit tourne un programme acceptant des ordres JTAG de relativement haut niveau (comme *ireg*, *dreg* ou *reset*). Ces derniers sont transmis à travers l'USB à partir d'une application tournant sous GNU/Linux. Celle-ci est aussi responsable de télécharger le programme embarqué dans le EZ-USB.

5.3 Principes du port JTAG

JTAG signifie Joint Test Action Group. Ce groupe a défini une norme de test de circuits intégrés. Ces tests s'effectuent par l'intermédiaire du TAP (Test Access Port) qui donne accès à un registre d'instruction (*ireg*) et plusieurs registres de donnée (*dreg*).

Quatre lignes donnent permettent de contrôler ce port :

TCK horloge donnée par le testeur,

TMS contrôle par le testeur de la machine d'état du TAP,

TDI donnée envoyée au circuit testé,

TDO donnée reçue du circuit testé.

Notre connecteur JTAG comprend deux lignes supplémentaires :

TRST* permet de redémarrer le TAP (pull-up),

RESET* ligne de RESET du CPU (pull-up).

Les registres JTAG sont des registres à décalage vers la gauche, dont le bit de poids faible est branché sur TDI, et le bit de poids fort sur TDO. Chaque accès à un registre est donc à la fois une lecture et une écriture.

La sélection des registres de donnée se fait par le chargement d'une instruction précise dans le registre d'instruction.

Les commandes JTAG du PXA250 sont décrites à la page 9-3 du manuel du PXA250 [3].

5.4 Initialisation du système

Le PXA250 possède 32k de cache d'instruction, plus un mini cache d'instruction de 2k. Ces deux caches sont modifiables par JTAG. Le mini cache est d'ailleurs inscriptible uniquement par JTAG. Il faut noter que le cache d'instruction ne se modifie pas en cas d'écriture de donnée à une adresse cachée. Le code auto-modifiable est donc à éviter. Lorsqu'un programme en charge un autre, il devra invalider le cache d'instruction.

Le PXA250 a également un cache de donnée de 32k, plus un mini-cache de 2k. Ces caches de données ne peuvent être utilisés que si la MMU est activée.

Pour pouvoir exécuter le programme de démarrage, les étapes suivantes sont nécessaires :

1. Maintenir le processeur dans l'état RESET,
2. Charger le programme de démarrage dans le cache d'instruction, et une routine d'interruption de debug dans le mini-cache d'instruction,
3. Laisser le processeur tourner en relâchant le RESET,
4. Le programme commencera à s'exécuter à l'adresse 0, il devra :
5. Initialiser les GPIO, le port série, et le contrôleur SDRAM,
6. Initialiser une pile en SDRAM.

A partir de ce moment là, un programme C ou C++ peut être exécuté. Celui-ci pourra par exemple écrire des informations provenant du port série ou du JTAG dans la FLASH.

Il est possible de modifier le cache d'instruction pendant que le processeur fonctionne, en évitant les conflits par logiciel, ce qui est impossible pour un premier démarrage. Activer la ligne RESET du processeur invalide l'entier du cache d'instruction. Il faut donc une méthode pour empêcher le CPU de démarrer pendant que la ligne RESET est désactivée pour pouvoir charger correctement le logiciel de démarrage. La une commande JTAG du PXA250 qui permet de maintenir le processeur dans un état interne RESET permet résoudre ce problème.

5.5 Charger le cache d'instruction par port JTAG

Le programme *jtag-usb* permet d'envoyer un programme dans le processeur à travers le port JTAG contrôlé par un microcontrôleur EZ-USB. Il fonctionne sous GNU/Linux et nécessite la bibliothèque *libusb* [17] pour être compilé.

Voici la liste des commandes du programme *jtag-usb* (appelée par la commande *help*) :

init - init JTAG connection with the EZ-USB microcontroller.

check - check connection with PXA250.

stop - stop CPU execution

loadic - load a file in the instruction cache

quit - quit this program

reset - reset the JTAG interface

boot - boot the CPU

help - list all commands or gives help about a command

Pour charger un programme dans la mémoire cache, il faut commencer par l'assembler, puis lancer le programme *jtag-usb* en exécutant les commandes suivantes :

1. init
2. stop

3. loadic fichier.bin 0x00000000
4. boot

Le PXA250 sautera alors à l'adresse du vecteur de démarrage, soit 0x00000000. Le cache d'instruction étant valide à cette adresse, le programme qui vient d'être chargé va s'exécuter.

Cette séquence de commandes est illustrée sur la figure 5.2.

5.6 Implémentation de *jtag-usb*

La figure 5.3 montre la structure générale de la partie logiciel de notre solution de démarrage. L'utilisateur donne des ordres via une ligne de commandes. Celle-ci ne fait que donner accès aux méthodes de la classe qui dirige le PXA250 comme décrit aux paragraphes 5.3 et 5.4.

L'exécution des commandes JTAG de base est assurée par une interface abstraite, de manière à pouvoir facilement adapter *jtag-usb* à l'utilisation d'une autre interface matérielle, par exemple le port parallèle.

Une classe, qui hérite d'un pilote pour microcontrôleur EzUSB, se charge ensuite de le contrôler en y chargeant un programme embarqué. Celui-ci traite les pattes du port JTAG, en obéissant aux ordres venant du PC.

5.7 Transformer *jtag-usb* en débogueur

Il serait possible de continuer le logiciel *jtag-usb* pour l'interfacer avec *gdb*, afin d'avoir une solution de *debug* complète.

Les capacités de *debug* du PXA250 via le port JTAG sont limitées. Le seul outil à disposition est un mini cache d'instruction de 2KO et un mini cache de donnée de 2KO également. Il faut un *debug handler* chargé par port JTAG dans ce mini cache avant le démarrage pour pouvoir examiner le contenu des registres, mettre des *breakpoints* ou effectuer d'autres opérations de *debug*.

Avec la coopération d'un tel *debug handler*, *jtag-usb* pourrait s'interfacer avec gdb, soit en modifiant gdb lui-même, soit en ouvrant un port TCP/IP sur lequel gdb se connectera. Cette coopération est illustrée sur la figure 5.4.

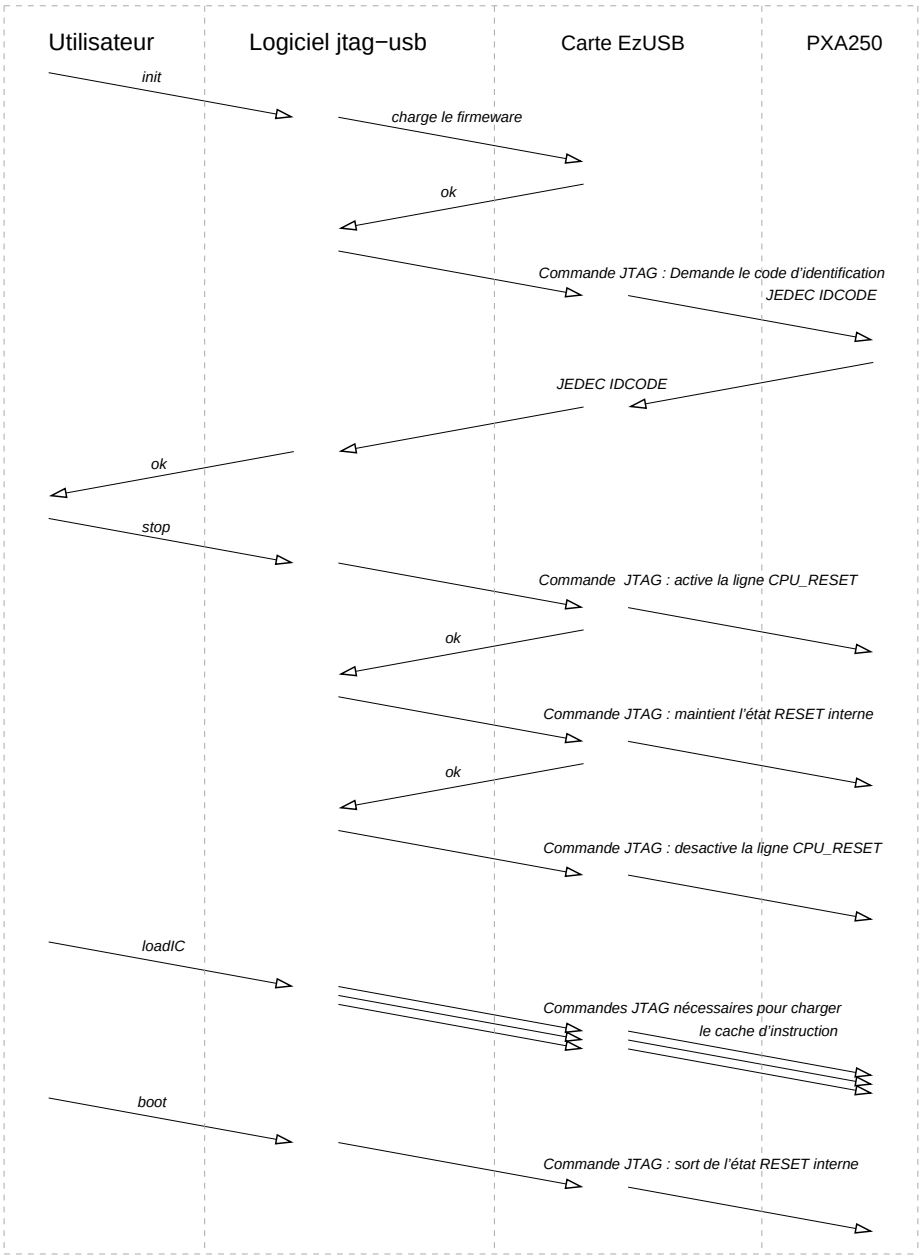

FIG. 5.2 – Chargement d'un logiciel dans le cache d'instruction du PXA250, en utilisant le programme *jtag-usb*.

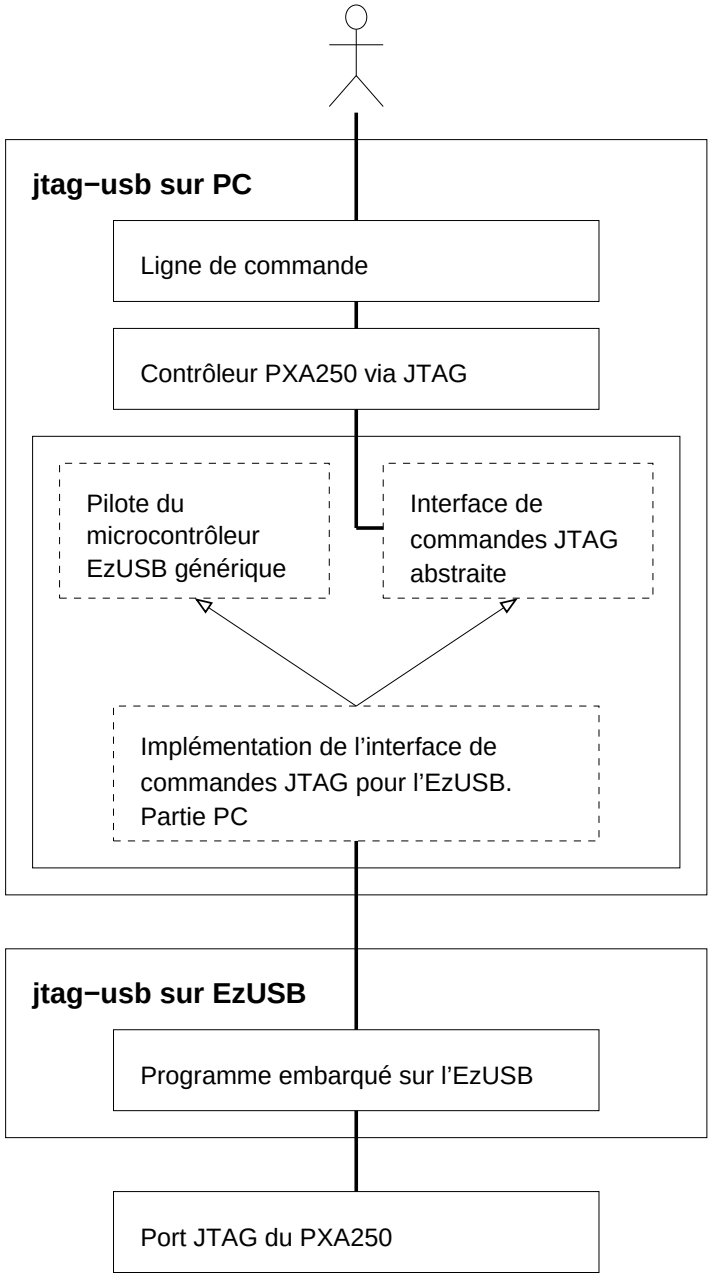

FIG. 5.3 – Structure du programme *jtag-usb*.

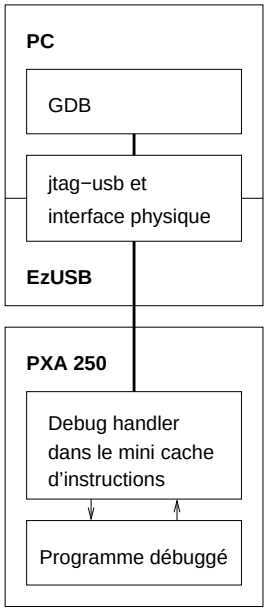

FIG. 5.4 – Le programme *jtag-usb* et un *debug handler* embarqué pourrait être interfacé avec gdb, pour lui permettre de déboguer via le port JTAG.

Chapitre 6

A propos de ce projet

6.1 Documentation en ligne

Nous avons réuni sur le web différents documents :
- Spécifications de tous les composants utilisés `http://in3www.epfl.ch/~smagnena/xscale/` ;
- documentation du programme jtag-usb `http://lappc22.epfl.ch/jtag-usb/` ;
- ce rapport `http://lappc22.epfl.ch/pxa250`.

6.2 Outils utilisés

La figure 6.1 liste les principaux logiciels utilisés pour ce projet. Nous avons choisi des solutions principalement *free software*, ceci pour plusieurs raisons :
- ces logiciels sont adaptés à l'architecture XScale ;
- leur code source est disponible ;
- ils sont gratuits et performants ;
- les logiciels libres correspondent à notre éthique ;
- GNU/Linux est plus pratique et plus résistant aux virus que Microsoft Windows.

6.3 Planning

Le tableau 6.2 montre le temps qu'ont pris les différentes tâches réalisées. Il faut cependant ajouter que nous avons commencé à rédiger le rapport dès le milieu du développement du schéma.

6.4 Échec, succès et travail non terminé

Au début de ce projet, beaucoup d'inconnues rendaient difficile la prévision de notre travail. *A posteriori*, nous constatons que le projet a bien avancé, dépassant nos espérances sur certains points, ne les satisfaisant pas sur d'autres. Le tableau 6.3 fait le point.

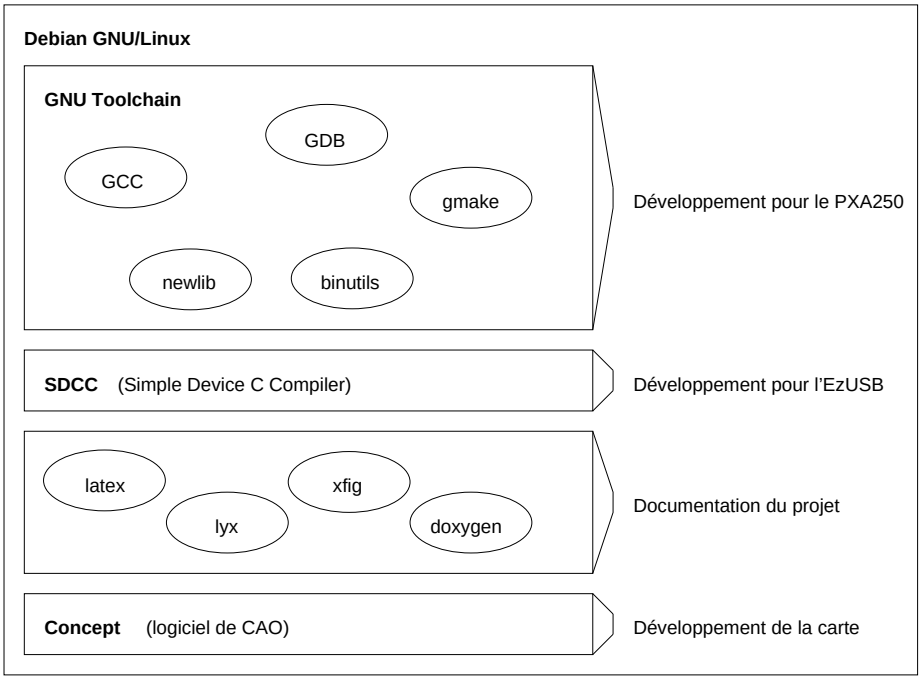

FIG. 6.1 – Liste des outils utilisés pour différentes tâches.

Semaine	Activité
1	documentation
2	documentation
3	documentation
4	schéma de la carte processeur
5	schéma de la carte processeur
6	schéma de la carte processeur / chaîne de développement
7	schéma de la carte processeur / chaîne de développement
8	schéma de la carte processeur / chaîne de développement
9	schéma de la carte d'alimentation / chaîne de développement
10	interface JTAG / initialisation du PXA250
11	interface JTAG / initialisation du PXA250
12	interface JTAG / rapport
13	interface JTAG / rapport
14	présentation

FIG. 6.2 – Organisation du temps de travail.

Tâche	Prévision	Status	Continuation
Réalisation de la carte	remplie	100%	test et debug
Outils de développement	partiellement remplie	80%	intégration gdb/*jtag-usb*
Système de démarrage	non prévu	100%	test, debug intégration gdb
Documentation	remplie	100%	site web
Carte d'alimentation de test	partiellement remplie	100%	-
Intégration robot	non prévu	20%	alimentation batterie moteurs

FIG. 6.3 – Évaluation des prévisions, statu et continuation des tâches effectuées.

Nous avons actuellement une carte montée non testée, et 5 PCB avec tous les composants nécessaires pour monter les cartes. Les tests et les corrections d'erreurs n'ont pas été prévus ce semestre, et nous n'avons effectivement pas eu le temps de les réaliser.

Par rapport aux buts du projet, il reste quelques tâches à compléter. Il faut tout d'abord tester la carte et corriger les éventuels bogues. Pour ce faire, il faut avoir des outils de développement fonctionnels (notamment *jtag-usb*). Or il n'est possible de tester ces derniers qu'avec une carte sous la main. Cette interdépendance rendra la phase de test assez complexe, et d'une durée indéterminée.

Nous avons tenté de concevoir une carte d'alimentation capable de s'interfacer avec le cyclope, moyennant une alimentation externe. Des erreurs – confusion entre un connecteur mâle et femelle, problèmes d'espaces – ont rendu cette carte inutilisable et nous l'avons transformée en carte de test, servant uniquement à brancher des alimentations de laboratoire.

L'intégration au robot nécessite une carte relativement complexe, avec des convertisseurs de tension. De plus, si la capacité du PXA250 à gérer une tension d'alimentation du coeur variable est présente, un contrôleur spécialisé ou une logique supplémentaire est requise. Ainsi, la partie intégration, bien que déjà entamée (schéma d'une carte avec convertisseurs A/D et drivers moteurs), demande encore un travail important pour supporter parfaitement le PXA250.

6.5 Évolutions possibles

Nous allons continuer ce projet pour notre travail de diplôme. Bien que le contenu exact de ce travail ne soit pas encore détaillé, nous devrons commencer par tester la carte, le système JTAG, et les outils de développement.

Nous aurons à écrire des logiciels de base, et nous allons probablement adapter GNU/Linux pour ce système. L'embarquement sur le robot cyclope nous demandera aussi du travail.

L'alternative possible du côté des outils de développement est la suivante : continuer à développer un système nous même, ou acheter une solution exis-

tante.

La conception de cartes périphériques peut également faire partie de notre projet de diplôme : une carte multimédia, par exemple, avec un écran LCD et un système audio. Une carte caméra est une autre possibilité, qui permettrait au PXA250 d'utiliser toute sa puissance pour le robot Cyclope.

Tous ces choix restent ouverts.

6.6 Conclusion

Ce projet a été très riche en enseignements pour nous. Il nous a appris à développer une carte, ce qui est une occasion unique de faire un peu de matériel dans un cursus d'informaticien très orienté logiciel. En chemin, nous avons pu expérimenter le protocole USB, comprendre le JTAG, voir les problèmes liés aux bootloaders, se familiariser avec la compilation d'un cross compilateur, subir une attaque de virus et installer GNU/Linux.

Bibliographie

[1] Intel XScale Technology http://www.intel.com/design/intelxscale/

[2] Intel PXA250 and PXA210 Applications Processors Design Guide http://in3www.epfl.ch/~smagnena/xscale/PXA250_Application_Processor_Design_Guide.pdf

[3] Intel XScale Microarchitecture for the PXA250 and PXA210 Applications Processors User's Manual http://in3www.epfl.ch/~smagnena/xscale/PXA250_users_manual.pdf

[4] Intel PXA250 and PXA210 Applications Processors Power Supply Design, Application Note, février 2002 http://in3www.epfl.ch/~smagnena/xscale/PXA250_Power_Supply_design_AppNote.pdf

[5] Intel PXA250 and PXA210 Applications Processors Power Fault Management, Application Note, mars 2002 http://in3www.epfl.ch/~smagnena/xscale/PXA250_Power_Fault_AppNote.pdf

[6] ARM Ltd Homepage http://www.arm.com/

[7] Cédric Gaudin, page personnelle. http://diwww.epfl.ch/~cgaudin

[8] Buffered H-Bridge. http://www.tavrasm.org/robot/vishay-si9986cy-2.pdf

[9] Maxim Semiconductor. Low-Power, 8-Channel, Serial 12-Bit ADC http://pdfserv.maxim-ic.com/arpdf/MAX1245.pdf

[10] Intel PXA250 Homepage http://www.intel.com/design/pca/prodbref/298620.htm

[11] Alliance Semiconductor Flash 8MBits http://www.gaw.ru/doc/Alliance/as29lv800.pdf

[12] Mosel Vitelic SDRAM chip 128 MBits http://www.moselvitelic.com/sec2/V54C3128(16-80-40)4V(T-S).pdf

[13] Maxim Semiconductor Reset circuit http://pdfserv.maxim-ic.com/arpdf/MAX803L-MAX810Z.pdf

[14] CompactFlash Association http://www.compactflash.org/

[15] USB.org http://www.usb.org/

[16] Cypress EZ-USB http://www.cypress.com/products/cat_beast_fifo.cfm?oid=112723&objectid=711EE37B-EDAF-4D81-831A1AA035DF12EB&foid=FC8F5931-5AFF-46AF-96FE79700B2CBB64

[17] libusb project home http://libusb.sourceforge.net/

[18] The GNU Toolchain for ARM targets HOWTO http://www.armlinux.org/docs/toolchain/toolchHOWTO.pdf

[19] High-speed BDM/JTAG Interface for GNU Debugger, Abatron http://www.abatron.ch/BDI/bdiGDB.html

[20] Architecture Reference Manual second edition, David Seal, Addison-Wesley