Introduction au Coldfire 5272

J.-M Friedt, S. Guinot, É. Carry
Association Projet Aurore, 16 route de Gray 25030 Besançon
http://projetaurore.assos.univ-fcomte.fr
28 avril 2005

Nous proposons ici une introduction pas à pas de la mise en œuvre d'un système embarqué autour du noyau uClinux. Il s'agit d'une mise en pratique visant à permettre de se lancer dans le monde du linux embarqué à moindres frais. Nous verrons dans un premier temps l'aspect matériel avec le circuit imprimé permettant de faire fonctionner le circuit de test. Puis nous présenterons l'environnement de travail et en particulier le cross-compilateur [1] pour générer un binaire Motorola sur un PC à base de processeur Intel. Enfin nous présenterons l'accès aux périphériques du processeur et le mode de programmation quelque peu différent sur système embarqué.

1 Le matériel

Le noyau uClinux [1, 2, 3] a été porté à une large gamme de microcontrôleurs et de processeurs. Sa principale contribution est de pouvoir se passer d'un gestionnaire de mémoire et donc de fonctionner sur des systèmes embarqués relativement simples ne comprenant qu'un microcontrôleur et quelques périphériques servant à le faire fonctionner. Nous nous somme fixés deux objectifs dans le choix du matériel : le coût et l'encombrement. La seconde contrainte tient à la nature de notre projet : le système final doit pouvoir être embarqué sur un système volant (d'où nécessité d'un poids faible, d'un encombrement réduit et d'une alimentation simple à mettre en œuvre). Notre choix s'est porté sur le uCdimm 5272 de Arcturus [4], répondant à notre critère de prix (275 dollars américains incluant les frais de port) et surtout étant le plus petit système que nous ayons trouvé pour une telle puissance de calcul (noter qu'une solution à base de microcontrôleur Renesas H8 [5], plus petite et moins chère, ne répondait pas à nos contraintes de puissance de calcul, notamment au niveau de la disponibilité d'une unité de calcul flottant). La carte sélectionnée comporte 4 MB de mémoire flash non-volatile, 8 MB de RAM, un processeur Motorola Coldfire cadencé à 66 MHz et un second port ethernet à 100 Mb/s (en plus du port équipant le processeur).

Le uCdimm 5272 vient sans documentation autre que les quelques pages disponibles sur le site web d'Arcturus (principalement connectique de la carte et commandes du bootloader se chargeant de l'initialisation du matériel). Une carte de développement beaucoup plus coûteuse est éventuellement disponible mais dépassait largement notre budget (uCevolution board). La datasheet du processeur embarqué, le Motorola Coldfire 5272 [6], devient rapidement une lecture fondamentale pour la compréhension des capacités de ce système tel que nous allons le décrire plus loin.

La connectique est surprenamment simple : la carte uCdimm 5272 n'a besoin que d'une alimentation 3,3 V (que nous fournissons par un régulateur de tension LM317 permettant une large gamme de tensions d'entrées – 5-37 V – et notamment l'alimentation du système par la tension lentement décroissante d'une batterie) convenablement découplée à la masse. Le régulateur de tension n'a même pas besoin de radiateur tellement la consommation est faible. Les ports RS232 sont directement utilisables : les convertisseurs de niveaux TTL vers RS232 sont embarqués sur la carte. De même pour les connecteurs ethernet : les transformateurs rendant les signaux de sorties différentielles sont embarqués et prennent une place conséquente sur le circuit imprimé. Le connecteur est un SODIMM 144 broches du type de ceux utilisés pour les mémoires d'ordinateur portable (acheté chez Digikey aux États Unis car introuvables en France [7]). Il faut cependant prévoir, pour palier au circuit de test commercialisé par Arcturus, la capacité de souder un connecteur CMS aux broches relativement serrées : un fer à souder pour CMS et une binoculaire sont indispensable pour ce travail en complément de la capacité de graver un circuit imprimé (simple face) avec des pistes espacées de 0,8 mm.

[1] cross-compilation : compilation d'un binaire à destination d'une architecture (Coldfire Motorola) sur un processeur d'une autre architecture (ici Intel x86 ou SPARC)

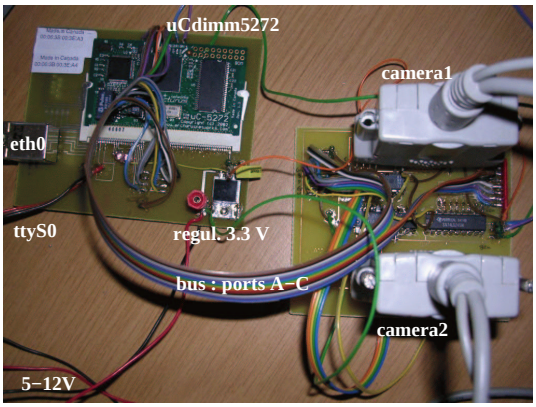

FIG. 1 – Circuit après l'émulation d'un port parallèle de PC compatible TTL et l'interfaçage de la webcam. Le circuit supportant la carte uCdimm (incorporant un régulateur de tension, un connecteur ethernet et un connecteur RS232) est visible à gauche tandis qu'une nappe transporte les bus A, B et C vers une carte fille chargée de l'interface CMOS-TTL et du multiplexage des bus vers les deux webcams.

2 Première mise sous tension

Les risques de mauvaise manipulation électrique sont relativement réduits du fait de la simplicité de la connectique. Le premier test est de vérifier les messages transmis par le uCdimm 5272 à la mise sous tension. Le terminal par défaut est le premier port série, selon un protocole à 9600 bauds et N81. Les messages suivants, obtenus soit au moyen de `cat < /dev/ttyS0` ou `cu -l /dev/ttyS0 -s 9600`, apparaissent dès la mise sous tension :

```
uCbootloader 1.7.7r1
(c) Copyright 2001-2004 Arcturus Networks Inc.
All Rights Reserved.

CACHE on
BOOT FLASH type 00c8 AT49BV32XA @0x10c00000
DP|002000 DP|004000 DP|006000 DP|008000 DP|00a000 DP|00c000
DP|00e000 DP|010000 DP|020000 D-|030000 D-|040000 D-|050000
D-|060000 D-|070000 D-|080000 D-|090000 D-|0a0000 D-|0b0000
D-|0c0000 D-|0d0000 D-|0e0000 D-|0f0000 D-|100000 D-|110000
D-|120000 D-|130000 D-|140000 D-|150000 D-|160000 D-|170000
D-|180000 D-|190000 D-|1a0000 D-|1b0000 D-|1c0000 D-|1d0000
D-|1e0000 D-|1f0000 D-|200000 D-|210000 D-|220000 D-|230000
D-|240000 D-|250000 D-|260000 D-|270000 D-|280000 D-|290000
D-|2a0000 D-|2b0000 D-|2c0000 D-|2d0000 D-|2e0000 D-|2f0000
...
D-|400000

B$
```

La commande "`go`" permet alors de lancer le système d'exploitation flashé en mémoire non-volatile. L'uCdimm 5272 vient d'origine avec un système uClinux fonctionnel. Une séquence de boot classique est amorcée pour s'achever sur une demande de login : le seul compte par défaut est `root` avec le mot de passe `uClinux`. La découverte des outils disponibles est rapide : tous les binaires sont concentrés dans `/bin`. Nous verrons plus loin, section 6, comment utiliser `minicom` pour communiquer et transférer des images de disque.

Ayant vérifié le bon fonctionnement du port série, il nous reste à vérifier la connexion ethernet. L'adresse IP fixe par défaut du uCdimm est 192.168.1.200, initialisée si, et seulement si, une connexion ethernet existe au boot. Les commandes de configuration du réseau `ifconfig` et `route` sont disponibles pour changer ces paramètres et un `ping` sur l'hôte servant de terminal sur port série permet de valider le fonctionnement de l'ethernet. Le système par défaut vient avec un daemon telnet (login : root, passwd : uClinux) en attente de connexion ainsi que `mount`. Le système de fichier de l'hôte peut donc être exporté par NFS avec la commande suivante : `mount -o nolock,moutvers=2 IP_hote :répertoire /mnt` où les options sont nécessaires pour la bonne communication entre le serveur NFS intégré dans un noyau 2.4.22 et l'uCdimm.

3 Cross-compilation

Le système de base étant fonctionnel, nous devons apprendre à cross-compiler nos propres applications pour

- optimiser le noyau aux applications prévues
- ajouter de nouvelles applications (par exemple serveur web, éditeur de texte, etc ...)
- développer nos propres applications

Nous avons recompilé la chaîne de développement à destination de m68k-elf (i.e. binaire pour Motorola 68k au format ELF qui sera ensuite transformé en flat binary format pour être directement exécutable sous uClinux) sur PC à base d'Intel et sur SPARC. La méthode la plus simple sur PC est de désarchiver une image du répertoire contenant les binaires précompilés pour processeur intel des outils de développement (`gcc`, `ld`, `as` et librairies associées) [8]. Sous SPARC la compilation de tous ces outils est nécessaire : `binutils` (v.2.10), `gcc` (v.2.95.3), `genromfs` (v.0.5.1) et `STLport` (v.4.5.3), avec l'option `TARGET=m68k-elf`, l'ensemble de la procédure étant gérée par le script `build-uclinux-tools.sh` [9].

Une fois les outils de compilation à notre disposition, il nous faut récupérer une archive de uClinux [10], la configurer pour notre matériel et compiler un nouveau noyau (voir section 5). C'est aussi au cours de cette compilation que nous compilerons notre application à embarquer tel que décrit dans `Documentation/Adding-User-Apps-HOTO` de la distribution uClinux.

4 Programmation sous uClinux

L'environnement de travail sur un système uClinux se rapproche plus de celui familier aux utilisateurs de microcontrôleurs ou de MS-DOS que d'un environnement de système multiutilisateurs et multitâches. L'accès à l'ensemble de la mémoire par notre programme est une propriété de l'absence de gestionnaire de mémoire (MMU) qui n'induira donc pas d'erreur de segment (`segmentation fault`) en cas d'accès à une zone mémoire non-dédiée au processus faisant la requête. Cette différence majeure va nous offrir la possibilité d'accéder à tous les périphériques en accédant à des emplacements mémoires prédéfinis puisque sur une architecture Motorola les accès aux périphériques d'entrée-sortie se font comme des accès mémoire. Ainsi nous utiliserons des lignes quelques peu inhabituelles où les valeurs de pointeurs sont définies pour pointer des zones mémoire prédéfinies.

Une premier exemple simple consiste à faire clignoter des diodes connectées aux ports A et C (broches 53-60 et 71-74 sur le uCdimm 5272). La lecture de la datasheet du Coldfire nous indique que ces ports se situent à des registres de base, `MBAR`, incrémentés de 0x86 et 0x96 respectivement. Le choix des ports A et C est dicté par le fait que les fonctions du port B sont multiplexées entre un port général numérique (GPIO – General Purpose IO) et le premier port série (ttyS0) dont nous voulons conserver l'accès.

Nous verrons plus loin [6, chapitre 17] que 3 registres contrôlent le comportement de chaque port :

- `PiCNT` (`i=A, B`) définit le mode de multiplexage de chaque port (par exemple le port B est multiplexé avec les premiers ports ethernet/série)
- `PiDDR` (`i=A, B, C`) définit la direction de chaque broche
- `PiDAT` (`i=A, B, C`) définit le niveau de chaque broche (en lecture si la broche est configurée en entrée)

5 Compiler son propre noyau

La distribution uClinux comprend les noyaux 2.4 et 2.6 adaptés aux systèmes sans MMU. La configuration se fait comme pour une compilation classique de linux : `make menuconfig`. La différence vient ensuite du menu : au lieu d'arriver directement sur le menu de configuration du noyau, l'utilisateur sélectionne le type de carte sur lequel se fait le développement (dans notre cas

Arcturus 5272) et choisit s'il veut modifier les paramètres du noyau (dans notre cas pour changer la fréquence d'horloge de 48 à 66 MHz et la quantité de RAM de 2 à 4 MB) et/ou les application compilées dans l'image de disque à charger en RAM. Nous utilisons comme librairie C compacte uClibc. En quittant ce premier menu les menus suivants (configuration noyau et utilitaires) apparaissent. La compilation est complétée par `make dep` et `make` qui place dans le répertoire `images` une image disque prête à être transférée en RAM du système embarqué `image.bin` et dans le répertoire `romfs` le contenu de cette image. En montant par NFS ce dernier répertoire, il est possible d'exécuter les nouveaux binaires sans avoir à transférer l'image complète en RAM par le port RS232 (commande `rx` du bootloader), opération relativement longue décrite dans le prochain paragraphe. Notons que le noyau s'exécute *toujours* de la RAM pour des raisons de temps d'accès aux différents types de mémoires : même un noyau destiné à être stocké en mémoire flash doit être défini (menu `Processor type and features` de la configuration du noyau) avec `Kernel executes from RAM`.

6 Transfert d'une image du système de fichier

Nous avons vu que `cat` ou `cu` permettent d'observer le comportement de l'uCdimm. Un outil plus souple et plus complet est `minicom`. Ce dernier nécessite un certain nombre de configurations pour être utilisable :

1. lancer le panneau de contrôle de `minicom` par "CTRL-A O"
2. Serial port setup → Serial Device → /dev/ttyS0
3. Serial port setup → Bps/Par/Bits → 9600 8N1
4. Serial port setup → Hardware Flow Control → No
5. Serial port setup → Software Flow Control → No

et vérifier que sous "'File transfer protocols" on ait "xmodem /usr/bin/sx -vv"

Il est inutile de tenter de transférer une image à 115200 bauds : alors que tous nos transferts ont fonctionné en 57600, ils ont toujours échoué sous 115200. Une fois l'image transférée, il faut penser à repasser le terminal en 9600 bauds avant de lancer `goram` puisque linux ouvre un terminal sur le port série à cette vitesse par défaut. Après avoir constaté que l'image placée en RAM se comporte comme prévu, elle peut être transférée de façon permanente en mémoire flash par l'instruction `program` du bootloader, à exécuter après un `rx`. Dans notre cas un résultat satisfaisant a été obtenu après :

- avoir édité `etc/inittab` et en avoir retiré les exécutions de `agetty` pour libérer les ports série (ceci dans le répertoire `vendor/Arcturus/uC5272` faute de quoi le fichier sera écrasé à la prochaine compilation du noyau)
- avoir ajouté dans `etc` un fichier `passwd` sans mot de passe pour root (dans le répertoire `romfs/etc` puisque ce fichier n'est pas remplacé à la compilation d'un nouveau noyau)
- avoir ajouté dans les applications utilisateurs des outils fondamentaux tels que `ls`, `mkdir` et, pour avoir NFS, `portmap` et `mount`. Noter que pour ce dernier il faut modifier la configuration par défaut de uClibc en activant les RPC (désactivés par défaut : modifier `UCLIBC_HAS_RPC=y`) en éditant le fichier `config.uClibc`
- avoir ajouté nos propres binaires (soit dans le répertoire `romfs/bin` après compilation manuelle, soit en ayant modifié de façon appropriée le `Makefile` tel que décrit dans la section 3).

7 Développement d'une application pratique : transfert d'images

L'application pratique que nous désirons développer en vue de transmettre des images depuis un ballon captif consiste à :

1. capturer des images et éventuellement les compresser

2. remplacer la connection ethernet par une connexion sans fil
3. transférer en temps réel les images par la connexion réseau

Le premier point sera dans un premier temps réalisé au moyen d'anciennes webcams Connectix Quickcam noir et blanc qui se connectent normalement sur un port parallèle de PC. Ce premier exercice nous familiarisera avec la programmation des ports A, B et C en entrée et sortie, ainsi qu'avec les aspects électroniques de l'interfaçage d'une logique CMOS 3,3 V (uCdimm) avec une logique TTL (5V, webcam).

Le second point sera simplifié à l'extrême en adoptant une solution commerciale : une interface ethernet-wifi se charge de la communication sans fil.

Le dernier point sera obtenu en modifiant le programme de contrôle de la caméra pour transférer les valeurs des pixels par réseau au lieu de sauvegarder dans un fichier.

7.1 Programmation de la webcam

Le protocole de communication entre le port parallèle d'un PC et la Quickcam n'est pas dans le domaine publique, mais plusieurs logiciels opensource (Linux, DOS et 68HC11 [11]) nous fournissent les étapes successives de configuration de la caméra puis de capture d'images.

Nous constatons qu'il nous faut 8 bits de communication du uCdimm vers la caméra (envoi des commandes : port A), 5 bits de la caméra vers le uCdimm (lecture des pixels par quartets et handshake : port B) et 2 bits de contrôle de l'uCdimm vers la caméra (port C). Nous utiliserons plus tard deux bits supplémentaires du port B en sortie pour le multiplexage des ports afin de connecter deux caméras simultanément. L'interfaçage entre les logiques 3,3 V et TTL se fait au moyen de résistances (pull-up de 3,3 V vers TTL, limitation de courant de TTL vers 3,3 V 2). Nous avons constaté que toutes les broches, tant en entrée qu'en sortie, nécessitent une logique d'interfaçage faute de quoi la communication avec la caméra ne se fait pas correctement. Le port A est donc connecté à la caméra via une interface de bus 74LS245 tandis que les ports B et C sont connectés à un latch 74HCT573 (le choix des composants étant simplement dicté par leur disponibilité, tout autre composant TTL pour interfaçage de bus pouvant faire l'affaire). L'avantage d'utiliser un 74245 pour interfacer le port A est que nous pourrons ultérieurement utiliser ce port en mode bidirectionnel (et ainsi accélérer le transfert des images entre la caméra et le uCdimm).

Le système d'exploitation uClinux nous fournit les emplacements d'un certain nombre de registres qui nous seront utiles : le registre d'adresse de base des registres de configuration MBAR est défini dans `asm/coldfire.h` (constante `MCF_MBAR`) tandis que les registres de configuration eux même sont définis dans `asm-m68knommu/m5272sim.h`.

Après avoir observé que les emplacements des registres tels que définis dans les entêtes de uClinux correspondent bien aux valeurs données dans la datasheet du Coldfire 5272 de Motorola, nous avons eu la surprise de constater que la programmation des registres A à C de la carte Arcturus ne semble pas suivre les mêmes règles de programmation que celles indiquées par Motorola. Il est en fait apparu que le port nommé port B sur l'uCdimm5272 est en fait le mot de poids faible du port C du Coldfire 5272, le port A de l'uCdimm est le poids faible du port C du Coldfire 5272, et finalement le port C de l'uCdimm est l'octet de poids fort du port A du Coldfire. Ce résultat surprenant se confirme par la manipulation des ports de direction : le mot de poids fort de PCDDR doit être manipulé pour rendre le port B de l'uCdimm sortant.

En résumé :

- les ports se configurent en mode GPIO par `*((volatile unsigned long *)(MCF_MBAR + MCFSIM_PACNT)) = 0x40000000;` pour utiliser les ports A et B comme port d'entrée/sortie généraux. Il est inutile de modifier `MCFSIM_PBCNT` avec la nomenclature que nous venons de décrire : les interfaces ethernet et RS232 restent disponibles.
- les registres de direction des ports PADDR et PCDDR définissent les directions des ports, l'octet de poids fort de PC (nomenclature Coldfire) correspondant à la direction du port B (nomenclature Arcturus Networks). Ainsi pour configure les ports A et C en sortie, les 5 bits de poids faible du port B en entrée et les 3 bits de poids fort en sortie, on utilisera

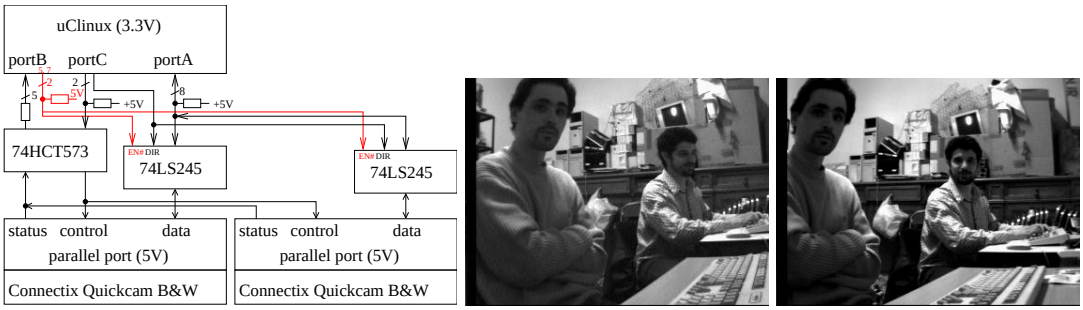

FIG. 2 – Gauche : circuit d'interfaçage entre l'uCdimm (logique CMOS 3,3 V) et une logique TTL compatible avec un port parallèle de PC pour connexion d'une webcam. Les composants TTL servent de tampon et éventuellement de multiplexeur entre le processeur et les périphériques. Le port A sert à la communication bidirectionnelle des données (commandes de l'uCdimm vers les caméras, valeurs des pixels dans l'autre sens), 5 bits de poids faible du port B servent de port de lecture des données issues des caméras, le port C sert à transmettre les signaux de contrôle à la caméra et finalement 2 bits de poids fort du port B servent à l'arbitrage du multiplexage des bus entre les deux caméras. Droite : exemple d'images stéréoscopiques obtenues au moyen des caméras connectées à l'uCdimm (6 bits/pixel).

```
*((volatile unsigned short *)(MCF_MBAR + MCFSIM_PADDR)) = 0x00ff ;
((volatile unsigned short *)(MCF_MBAR + MCFSIM_PCDDR)) = 0xe0ff ;
```

– l'écriture sur les ports se fait alors de la façon suivante :
`*((volatile unsigned char *)(MCF_MBAR+MCFSIM_PADAT+1))=valeur ;`
pour écrire sur le *port C* et
`*((volatile unsigned char *)(MCF_MBAR+MCFSIM_PCDAT+1))=valeur ;`
pour écrire sur le *port A* (noter que les noms du port et du registre sont inversés dans ces cas du fait des assignations différents de noms entre Arcturus Networks et Motorola).
La lecture sur le port B se fait par
`valeur=*((volatile unsigned char*)(MCF_MBAR + MCFSIM_PCDAT )) ;`

Une fois équipés de ces fonctions de base (tant matérielles que logicielles), il ne nous reste plus qu'à implémenter les protocoles de communication pour obtenir une image telle que présentée à la figure 2 (droite). Les images sont acquises en boucle et dans un premier temps stockées sur la partition NFS pour visualisation ultérieure sur un PC équipé d'une interface graphique.

Programme de contrôle de la Quickcam

Le programme tel que décrit ici [12] permet de capturer périodiquement une image sur l'Cdimm et de la transférer par réseau à des clients tournant sur PC pour affichage et sauvegarde. Ainsi, nous avons sélectionné de placer le serveur multithreadé sur l'uCdimm (utilisation de la librairie pthread optimisée pour les applications embarquées, fournie avec uClinux) de façon à permettre à une multitude de PCs distants de se connecter directement au serveur d'image, notamment avec l'objectif d'une capture d'images stéréoscopiques par un uCdimm embarqué (ballon sonde ou drone par exemple) et de permettre à de nombreux clients de visualiser simultanément le résultat de ces acquisitions. La connection ethernet a été remplacée de façon transparente par une connection sans-fil Wifi au moyen d'un convertisseur ethernet-wifi DLink-DWL810+, compatible en terme de tension d'alimentation avec une application embarquée (5 V régulés par un 7805, 600 mA).

Nous avons complété l'électronique d'interfaçage en doublant tous les composants de gestion du bus (75245 et 74573) et ainsi pu connecter deux caméras pour des applications de mesure de distance d'objets par stéréoscopie, la puissance de calcul du Coldfire 5272 étant suffisante pour effectuer un calcul d'intercorrélation sur certaines lignes des images acquises. De plus, nous avons implémenté la compression jpeg des images soit avant transmission (pour libérer de la bande

passante et ainsi augmenter la portée du Wifi), soit après réception d'une image pour affichage direct sur une interface web.

8 Utilisation des PWM

Les générateurs d'impulsions modulées en largeur (PWM) sont des périphériques pratiques pour le contrôle de la vitesse de moteurs DC, pour réaliser des convertisseurs numériques-analogiques (après lissage par un filtre passe-bas) ou, dans notre cas, le contrôle angulaire de servo-moteurs de modélisme.

Deux des PWMs intégrées dans le Coldfire 5272 sont mises à disposition de l'utilisateur sur l'uCdimm. Leur utilisation est conforme à la description fournie dans la datasheet du 5272 [6, chapitre 18] :

1. la PWM i est activée en mettant à 1 le bit de poids le plus fort du registre de contrôle `MCFSIM_PWCR`i à l'adresse `MBAR+0xc0+4*`i ($i = 0..2$)
2. la fréquence de répétition des pulses est déterminée par le facteur de division défini par les 4 bits de poids les plus faibles de ce même registre `MCFSIM_PWCR`i
3. le rapport cyclique est déterminé par la valeur placée dans `MCFSIM_PWWD`i à l'adresse `MBAR+0xd0+4*`i

Nous générons ainsi avec un facteur de division de 0x0c sur la PWM0 des pulses répétés toutes les 15,8 ms, de largeur 1,5 à 2,5 ms pour des valeurs de délais comprises entre 13 et 37 pour le contrôle des servo moteurs.

Nous voyons dans cet exemple la complémentarité des méthodes de développement sous uClinux : à la fois programmation sur un système d'exploitation de haut niveau (par exemple pour l'accès aux ports RS232 qui se fait sur l'uCdimm de façon strictement identique à la méthode employée sur PC) et de très bas niveau tel que nous le ferions sur un microcontrôleur en écrivant directement dans les registres de configuration matériel du processeur Coldfire 5272.

9 Gestion des interruptions

Les interruptions matérielles, qui déclenchent un événement logiciel associé à un événement matériel sur une des broches du processeur, présentent un cas particulier car seul le noyau peut y accéder. Il nous faut donc, pour utiliser cette fonctionnalité du processeur Coldfire, écrire un module qui sera inclus dans le noyau. Là encore nous allons trouver une structure de programme familière aux développeurs sous linux, avec des structures standards pour un module [13], et des spécificités du matériel qui imposent la lecture de la documentation du processeur [6, chapitre 7].

Dans un premier temps un ajout matériel s'impose pour déclencher une interruption matérielle au moyen d'un bouton poussoir et empêcher l'effet de rebond des lames souples d'un bouton (*debounce*) : nous ajoutons un trigger de schmitt 7414 avec un circuit RC de constante de temps faible (pour ne pas pénaliser la réactivité du circuit) afin de ne déclencher qu'une seule (et non plusieurs) interruption par appui de l'interrupteur. Le circuit, utilisant un composant alimenté en 3,3 V (type LVC), se connecte directement à la broche 77 du circuit uCdimm. En l'absence d'un composant compatible 3,3 V, il devrait être possible d'utiliser n'importe quelle version de composant TTL (5 V) connecté à la broche de déclenchement d'interruption matérielle via une résistance (fig. 3).

À même titre que pour l'activation des interruptions sur PC compatible IBM, la séquence d'initialisation et d'acquittement après traitement de l'interruption est fortement dépendante du matériel. Dans le cas particulier qui nous intéresse ici, à savoir la gestion de l'interruption IRQ0# dans la nomenclature Arcturus (qui correspond à l'interruption INT2 dans la nomenclature Motorola), le registre sur lequel nous devrons agir est ICR1 à l'adresse MBAR+0x20. L'initialisation d'une interruption i se fait en plaçant la valeur 0x111 dans INTiIPL. L'acquittement d'une interruption après gestion de l'évènement se fait en plaçant la valeur 1 dans le bit INTiPI. Il faut

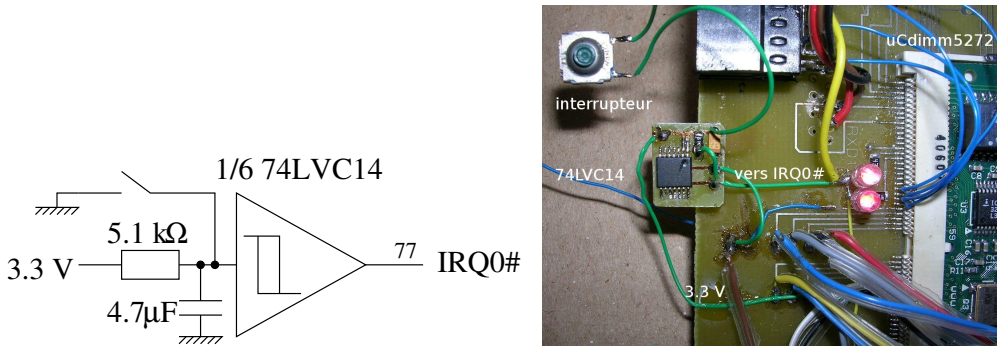

FIG. 3 – Schéma de principe (gauche) et implémentation (droite) du circuit de debounce pour déclencher une interruption matérielle au moyen d'un bouton-poussoir.

penser, au cours de la gestion de l'interruption (fonction `uc_int_irqhandler()`), de désactiver l'interruption avant de l'acquitter, puis effectuer les opérations associées à cette interruption, avant de finalement réactiver l'interruption à la fin de cette procédure. Toutes les autres fonctionnalités consistent en de la programmation de module noyau standard tel que présenté dans l'exemple de programme ci-dessous, où nous n'avons conservé que les parties du modules spécifiques à la gestion des interruptions par le noyau et le processeur Coldfire, ainsi que la communication avec l'espace utilisateur. Une version complète et fonctionnelle du module est disponibles à [12].

```
#define IRQ_DEFAULT 66
static int uc_int_major = 6;

static struct file_operations uc_int_fops =
{owner: THIS_MODULE,
 read: uc_int_read,
 open: uc_int_open,
 release: uc_int_release,
};

static void uc_int_irqhandler (int irq, void *dev_id, struct pt_regs *regs)
{unsigned long tmp;
 [...]
 tmp = *((volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1));

 /* disable interrupts */
 tmp &=0xf8ffffff;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 /* we ack this interruption to the hardware */
 tmp |=0x08000000;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 /* and wake up user application waiting for a read */
 wake_up_interruptible (&dev->uc_int_queue);
 /* restore interrupts */
 tmp |= 0x0f000000;*(volatile unsigned long *)(MCF_MBAR+MCFSIM_ICR1)=tmp;
 [...]
}
```

```
static int uc_int_open (struct inode *inode, struct file *file)
{[...]
 tmp = *((volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1));
 tmp |= 0x0f000000;
 *(volatile unsigned long *) (MCF_MBAR + MCFSIM_ICR1) = tmp;
 if ((retval = request_irq (uc_int_irq, uc_int_irqhandler, SA_INTERRUPT, "uc_int", dev)))
    {dbg("unable to assign irq %d", uc_int_irq);goto exit_sem;}
 [...]
}

static ssize_t uc_int_read (struct file *file, char *buffer, size_t count, loff_t *ppos)
{unsigned long tmp[4];
 [...]
 interruptible_sleep_on (&dev->uc_int_queue); /* sleep until a interrupt occurs */
 memcpy ((void *) tmp, (void *) (MCF_MBAR + MCFSIM_ICR1), sizeof (tmp));
 [...]
}

/* the realease method is call when user want close () the device */
static int uc_int_release (struct inode *inode, struct file *file)
{[...]
 free_irq (dev->uc_int_irq, NULL);
 [...]
}
```

Ce module enregistre la requête pour l'INT2 (nomenclature Motorola) au moyen de la fonction `request_irq()`. Un client se met alors à l'écoute d'une interface (`/dev/uc_int` défini avec un major number de 6 au moyen de `mknod uc_int c 6 0` dans `romfs/dev` du répertoire d'installation de uClinux) : la fonction `read()` étant bloquante, l'exécution du client ne continuera que lors du déclenchement de l'interruption matérielle. Le module transfert alors au client 16 octets correspondant aux valeurs contenues dans les registres INTiPL, $i \in [1..4]$ (i.e. mot de poids fort du registre ICR1).

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <asm/coldfire.h>           // defines MCF_MBAR
#include <asm-m68knommu/m5272sim.h> // defines PADDR

int main(int argc,char **argv)
{char i[16];int fd;char n;
```

```
 fd=open("/dev/uc_int",O_RDONLY);

 while (1) {read(fd,i,16);
 for (n=0;n<16;n++) printf("%x ",((char)(i[n]))&0xff);
 printf("\n");
        }
 return(0);
}
```

La méthode de compilation de ce module est décrite dans le document disponible à `linux-2.4.x/Documentation/kbuild` dans l'arborescence uClinux (en particulier `config-language.txt`

et `makefiles.txt` indiquent les modifications à apporter à `Config.in` et au `Makefile` pour inclure le nouveau module dans la séquence de compilation). En résumé, nous créons un sous-répertoire nommé `uc_int` dans le répertoire `linux-2.4.x/drivers/char` de uClinux, dans lequel nous plaçons le code source de notre module et un Makefile incluant les lignes `obj-$(CONFIG_UC_INT) := uc_int.o` et `include $(TOPDIR)/Rules.make`. Puis dans le Makefile du répertoire `char` nous ajoutons

```
ifeq ($(CONFIG_UC_INT),m)
    subdir-$(CONFIG_UC_INT) += uc_int
    obj-$(CONFIG_UC_INT) += uc_int/uc_int.o
endif
```

et enfin dans `linux-2.4.x/drivers/char/Config.in` nous ajoutons `tristate 'uc_int driver' CONFIG_UC_INT` pour créer une nouvelle entrée dans le menu de configuration du noyau. Un `make` du noyau créera alors le module `uc_int.o` qui sera inclus dans le noyau par un `insmod` tel qu'effectué classiquement sous linux.

10 Résultats

Nous avons embarqué notre montage incluant la carte uCdimm5272, les deux webcams noir et blanc présentées ci-dessus orientables par un servo moteur (contrôlé par la PWM), une appareil photo numérique Olympus C-860L (commandable par une connexion RS232 [14]) et un module wifi DLink DWL-810+ dans une nacelle fixée à un ballon captif de 2,60 m de diamètre gonflé à l'helium (European Balloon Corporation, Belgique) [15]. La liaison wifi pour transmettre au sol les image des webcams et commander l'appareil photo numérique fonctionne en l'absence d'obstacles à plus de 150 m (Fig. 4). L'absence de convertisseur analogique-numérique pour une acquisition de télémesure est compensée par la connexion sur le second port RS232 de l'uCdimm d'un microcontrôleur (ADuC814 [16]) chargé de lire une valeur analogique (dans notre cas la température de la nacelle) et de stocker le résultat dans un fichier monté par NFS à un serveur au sol. L'ensemble de ces dispositifs est alimenté par une batterie Lithium-polymère (T2M Powerhouse) de capacité 2000 mA.h pour une autonomie totale d'environ 1 h. L'ensemble de la nacelle pèse de l'orde de 1 kg.

11 Conclusion

Nous avons présenté les étapes successives de développement d'une application embarquée – acquisition et transfert d'images depuis un ballon captif – à base d'uClinux. Nous nous sommes dans un premier temps familiarisé avec le mode de développement (bootloader, cross-compilation et transfert des programmes résultants par NFS) avant d'aborder les aspects matériels de l'interfaçage et l'accès logiciel aux ports d'entrée-sortie. L'intérêt majeur de cartes du type de l'uCdimm 5272 de Arcturus Networks est la disponibilité à coût et consommation électrique réduits, d'une puissance de calcul importante et de la disponibilité de toute l'infrastructure de développement et des librairies linux.

Plusieurs constructeurs mettent sur le marché de nouvelles cartes aux fonctionnalités nouvelles et des facteurs de forme de plus en plus réduits ouvrant la perspective de nouvelles applications à peine imaginables il y a quelques années : nous venons de faire l'acquisition d'une carte DIL/NetPC DNP/5280 [18] dont une présentation complète fera l'objet d'un article ultérieur : ce circuit est basé sur le Coldfire 5282 qui offre de nombreux périphériques très utiles et indisponibles sur le 5272 – par exemple des convertisseurs analogique-numérique, un bus I^2C ou un bus CAN – ainsi qu'un support considérablement plus simple à mettre en œuvre (DIL40). Malgré la ressemblance des nomenclatures, le modèle de programmation est totalement différent du 5272 puisque comme sur les ordinateurs équipés de MMU, il faut passer par le noyau pour tous les accès aux registres associés à des fonctions matérielles du 5282.

FIG. 4 – Haut : photographie aérienne du parc de l'observatoire de Besançon (gauche) et coucher de soleil sur l'École Nationale Supérieur de Mécanique et des Microtechiques (ENSMM), la nouvelle Maison des Microtechniques en cours de construction et les bâtiments du CROUS (droite). Bas : prises de vue du Laboratoire de Physique et Métrologie des Oscillateurs (FEMTO-ST/LPMO) de Besançon, et image stéréoscopique résultante destinée à être visualisée au moyen de lunettes dont un verre est teinté en bleu et l'autre en rouge [17].

Remerciements

Nous remercions F. Vernotte, directeur de l'Observatoire de Besançon, pour son hébergement gracieux de l'association étudiante Projet Aurore dans ses locaux, ainsi que H. Bässmann (The Imaging Source, http://www.theimagingsource.com/) pour nous avoir offert deux optiques de webcam identiques (ref. Adogon L12mf3.6f) pour nous permettre de réaliser le montage d'imagerie stéréoscopique. Vincent Giordanno et l'équipe "métrologie des oscillateurs" du laboratoire FEMTO-ST/LPMO nous ont fourni l'helium pour la ballon captif. L'association de diffusion des logiciels libres sur la Franche-Comté – Sequanux (www.sequanux.org) – est remerciée pour son support logistique.

Références

[1] "Linux embarqué : le projet uClinux", Linux Magazine, pp.16-22 (Février 2002)

[2] "Linux et le système sur silicium", Linux Magazine, pp.50-59 (Avril 2005)

[3] M. Opdenacker, "Embedded Linux kernel and driver development", disponible à http://free-electrons.com

[4] http://www.arcturusnetworks.com/coldfire5272.shtml

[5] http://friedtj.free.fr/h8eng.pdf

[6] MCF5272 ColdFire Integrated Microprocessor User Manual (MCF5272UM), disponible à http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCF5272

[7] http://www.digikey.com, référence 54697-1440

[8] http://www.uclinux.org/pub/uClinux/uclinux-elf-tools/m68k-elf-tools-20030314.sh

[9] http://www.uclinux.org/pub/uClinux/uclinux-elf-tools/gcc-3/

[10] http://www.uclinux.org/pub/uClinux/dist/

[11] http://www.seattlerobotics.org/encoder/200009/qcam.html

[12] http://projets.sequanux.org/membres/sim/uclinux-webcam/

[13] J. Corbet, A. Rubini & G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly Ed. (2005)

[14] http://photopc.sourceforge.net/protocol.html

[15] http://projetaurore.assos.univ-fcomte.fr/ballon/

[16] http://friedtj.free.fr/aduc816.pdf

[17] W.R. Newcott, *Return to Mars* et J.B. MacInnis, E. Kristof ,*Titanic : Tragedy in Three Dimensions*, National Geographic, Aout 1998

[18] http://www.dilnetpc.com/dnp0038.htm