Mise à jour du contrôleur du robot bipède BIP

FUCHET Jérôme

août 2003

Table des matières

Table des figures

Chapitre 1

Introduction

Ce document présente le travail que j'ai réalisé pour la mise à jour de la carte processeur du contrôleur du robot bipède BIP. Le contrôleur du robot BIP récupère les données en provenance des capteurs, calcule des consignes à l'aide d'une loi de commande et envoie les consignes aux moteurs des jambes. La première version du contôleur était composée d'une carte processeur Motorola MVME162 sous le système d'exploitation temps réel VxWorks et de modules IP d'entrée/sortie.
La mise à jour de ce contrôleur a consisté à changer la carte processeur. Ainsi l'équipe SED a fait l'acquisition d'une carte processeur PowerPC auprès de la société ACTIS[1]. La mise à jour consiste également à changer le système d'exploitation utilisé sur le contrôleur. Les logiciels doivent fonctionner sous les systèmes d'exploitation VxWorks, pour conserver la compatibilité avec l'ancien matériel, mais aussi sous Linux et Linux-RTAI[2]. Le reste du contrôleur, c'est-à-dire les entrées/sorties, reste inchangé.
Dans un premier temps je décris tous les détails qui concernent plus particulièrement l'utilisateur. Ensuite je présente une partie plus destinée aux développeurs, avec des détails de l'implémentation.
Ce document est un rapport technique mais j'ai également rédigé un rapport de stage[3]. Ce deux documents ne contiennent pas les mêmes informations. Aussi, si des élements viennent à vous manquer ou si certains détails ne sont pas assez clairs, la lecture du rapport de stage pourrait vous éclairer.

Chapitre 2

Manuel de l'utilisateur

Ce chapitre vous permet d'obtenir une carte processeur fonctionnelle et prête pour exécuter les programmes du robot bipède BIP.
Les différents sous-chapitres sont classés par ordre d'exécution : du boot loader à la tâche générée par ORCCAD[7].

2.1 La carte processeur PowerPC d'ACTIS

Il s'agit d'une carte processeur équipée du processeur Motorola PowerPC MPC8260[8]. La fréquence du processeur est de 200MHz et il est accompagné de 32Mo de mémoire vive et d'au moins 8Mo de mémoire Flash. Elle propose également une interface VME et deux interfaces Ethernet 100Mbits/s. Pour plus de détails vous pouvez vous référer au site Internet d'ACTIS[1] computer donné en bibliographie ou consulter le manuel utilisateur[4] livré avec la carte.
La carte est également livrée avec une offre logicielle sur CD. Elle comporte tous les outils de compilation croisée nécessaires au développement sur la carte processeur. Elle contient aussi les sources du Linux utilisé sur la carte. Le contenu du CD et les informations pour l'installation de l'environnement de travail sont dans le guide d'installation[5] qui accompagne la carte processeur.
Pour utiliser la carte il faut configurer l'environnement de travail, et surtout le serveur TFTP (voir le guide d'installation), une alimentation pour bus VME et une console série connectée à la carte processeur.
Tous les fichiers chargés par TFTP sont dans le répertoire `/tftpboot` de la machine configurée en serveur TFTP.

2.2 Le boot loader

2.2.1 Historique

Un historique est nécessaire car deux bootloaders sont disponibles sur la carte PowerPC. Je vais donc expliquer pourquoi cette paire alors qu'un seul bootloader pourrait être suffisant.
La carte PowerPC ACTIS VSBC6862 est livrée avec un bootloader dont le nom est ECMON. Je l'ai utilisé durant presque toute la durée du développement. Seulement lorsque j'ai voulu booter automatiquement les noyaux Linux et RTAI, j'ai découvert

que la version d'ECMON livrée n'était pas une version complète : certaines des possibilités offertes par la version disponible auprès d'ECRIN Systems ont été enlevées. Parmis ces fonctions supprimées figure celle du boot automatique. J'ai donc récupéré sur internet un bootloader GPL nommé u-boot qui est capable de booter les cartes à base de processeur PowerPC. Ce bootloader offre une fonction de boot automatique.

2.2.2 ECMON

ECMON, est le bootloader livré avec la carte PowerPC d'ACTIS : c'est un programme crée par la société ECRIN Systems[9]. Je présente ici les commandes qui peuvent se révéler utiles. Pour voir les autres commandes d'ECMON vous pouvez consulter le guide utilisateur [4] fourni par ACTIS avec sa carte.
Pour que ECMON s'exécute, il faut que le jumper J2-position1 (voir le manuel utilisateur livré avec la carte, chapitre 3.4) soit positionné.

Lancer le noyau Linux ou RTAI

Tout d'abord, assurez-vous que vous disposez du noyau à charger sur la carte. Normalement, il doit s'appeler `zImage.vsbc6862` et être dans le répertoire `/tftpboot` du serveur TFTP. Ensuite entrez les commandes suivantes sur le terminal série relié à la carte :

Désactive l'auto-négociation pour le port Ethernet 1 car elle ne fonctionne pas avec le routeur de la halle robotique
```
ECmon> mii 0 0 2000
```

Charge le noyau dont le nom est zImage.vsbc6862 par tftpboot à l'adresse 0x10000 par le port Ethernet 1
```
ECmon> nload 0 10000 zImage.vsbc6862
```

Lance le noyau
```
ECmon> go 10000
```

Flasher un programme

Il est possible grâce à ECMON de programmer la mémoire Flash présente sur la carte. J'ai utilisé cette fonctionnalité pour mettre le noyau Linux ou encore u-boot sur la carte.
ATTENTION : à l'adresse où vous écrirez votre programme. Choisissez-la assez grande pour ne pas écraser ECMON. Il faut écrire après les premiers 128ko du banc de Flash, donc après l'offset 0x20000.
De même n'écrivez pas votre programme au début du deuxième banc de Flash sous peine d'écraser u-boot. De manière plus générale, écrivez vos programmes après les 500 premiers kilo-octets des bancs de Flash.
Il faut dans un premier temps vous assurer que tftpboot est capable de trouver le fichier que vous souhaitez flasher. Ensuite entrez les commandes suivantes :

Désactive l'auto-négociation sur le port Ethernet 1 si nécessaire
```
ECmon> mii 0 0 2000
```

***Charge le fichier en mémoire* à l'adresse 0x10000**
```
ECmon> nload 0 10000 nomDuFichier
```

Efface la mémoire Flash là où vous voulez écrire votre programme
```
ECmon> eflash adresseMemoire tailleDuFichierEnHexa
```

Ecrit le fichier en mémoire Flash
```
ECmon> pflash 10000 tailleDuFichierEnHexa adresseMemoire
```

Flasher le noyau Linux ou le noyau RTAI

Il suffit de suivre la même méthode que pour flasher un programme en choisissant le nom de l'image du noyau comme nom de fichier.

Flasher u-boot

La procédure est identique à celle pour flasher un programme mais je vais la détailler car elle est plus sensible. En effet, il est très simple d'écraser ECMON lors de cette opération : u-boot est écrit au début du banc de Flash, il suffit donc de se tromper de banc.

Chargez l'image de u-boot, normalement elle s'appelle u-boot.bin
Effacez le début du deuxième banc de Flash :
```
ECmon> eflash fc000000 tailleDeu-boot.binEnHexa
```

Copiez l'image de u-boot en Flash :
```
ECmon> pflash 10000 tailleDeu-boot.binEnHexa fc000000
```

Si à présent vous éteignez la carte, que vous enlevez le jumper de P2-position1, et que vous redémarrez la carte vous devez voir u-boot s'exécuter.

2.2.3 u-boot

u-boot est un bootloader GPL que j'ai récupéré sur Internet. Il est installé sur le deuxième banc de Flash et pour être exécuté au démarrage de la carte, il faut que le jumper J2-position1 soit enlevé.

Compilation de u-boot

Les sources de u-boot sont dans le répertoire
`$ROBOTIQUE/PowerPC/u-boot/u-boot-0.4.0`.
Toutes les instructions pour la compilation et des informations supplémentaires sont dans le fichier `README.SED` dans les sources de u-boot.
Pour compiler le noyau :
Allez dans le répertoire des sources.

Positionnez la variable d'environnement CROSS_COMPILE avec le prefix des outils de compilation croisée :
```
$ setenv CROSS_COMPILE $ROBOTIQUE/PowerPC/opt/ecrin/crossdev/-
host/powerpc/bin/powerpc-linux-
```

Nettoyez les sources :
```
$ make mrproper
```

Configurez la compilation pour la carte ACTIS VSBC6862 :
```
$ make vsbc-6862_config
```

Compilez les sources :
```
$ make
```

Installez l'image de u-boot ainsi créée dans */tftpboot* :
```
$ make install
```

Préparer le noyau à lancer

u-boot ne peut pas utiliser directement l'image produite par la compilation du noyau. Il faut donc utiliser un programme fournit avec u-boot pour modifier le noyau.
Nous allons utiliser des makefiles pour effectuer la conversion. Ces makefile sont :
`$ROBOTIQUE/PowerPC/RTAI/ACTIS-2.4.18/make_vsbcImage` et
`$ROBOTIQUE/PowerPC/RTAI/make_uImage`.
Si vous souhaitez convertir un noyau Linux "normal" il vous faudra copier ces deux makefiles en respectant la hiérarchie utilisée pour la varsion RTAI. C'est-à-dire que `make_uImage` est dans le répertoire du dessus comparé à `make_vsbcImage`. Le résultat de la conversion portera le nom `uImage.vsbc6862` et sera crée dans le répertoire `/tftpboot`.

Allez dans le répertoire où se trouve `make_vsbcImage`
(`$ROBOTIQUE/PowerPC/RTAI/ACTIS-2.4.18` dans le cas de RTAI).

Tapez les commandes suivantes :
```
make image
```
Si votre noyau source est un noyau Linux tapez :
```
$ make UBootLinux
```

Si votre noyau source est un noyau RTAI tapez :
```
$ make UBootRTAI
```

Pour avoir l'aide

Pour connaître la liste des commandes reconnues tapez `help` ou `?`. Pour avoir une aide sur une commande précise taper `? nomDeLaCommande`.

Les variables d'environnement

u-boot vous permet d'enregistrer des variables d'environnement.
Il est possible de voir leur valeur avec la commande `printenv`. Voyons plus en détail la signification de quelques-une de ces variables :

`bootcmd`	est la commande exécutée à l'autoboot
`bootdelay`	est le temps au bout duquel l'autoboot entre en action
`ethaddr`	est l'adresse Ethernet de la carte, cette adresse est utilisée lors d'un tftpboot
`serverip`	est la'adresse ip du server des fichiers chargés par tftpboot
`ipaddr`	est l'adresse IP de la carte
`reprog`	est la commande d'auto-reprogrammation de u-boot
`bootfile`	est le nom du fichier chargé par défaut par tftpboot c'est donc lui qui est chargé lors de l'autoboot.

Par défaut les valeurs de ces variables sont les suivantes :

`bootcmd`	=	version ; echo ; tftpboot ; setenv bootargs root=/dev/nfs rw ip=auto ; bootm
`bootdelay`	=	5
`ethaddr`	=	00 :20 :15 :51 :00 :00
`serverip`	=	194.199.21.246
`ipaddr`	=	194.199.21.248
`reprog`	=	tftpboot 0x400000 /tftpboot/u-boot.bin ; protect off 1 :0-2 ; erase 1 :0-2 ; cp.b 400000 FE000000 $(filesize) ; protect on 1 :0-2
`bootfile`	=	uImage.vsbc6862

Il est possible de créer, de modifier ou de supprimer une variable d'environnement à l'aide de la commande `setenv`.
Le contenu des variables d'environnement peut être exécuté avec la commande `run nomDeLaVariable`.
Enfin, il est possible de sauver la valeurs des variables d'environnement pour les retrouver lors du reboot suivant avec la commande `saveenv`.

La mémoire Flash

L'état de la mémoire Flash peut être consulté à l'aide de la commande `fli`. Il est ainsi possible de voir quels sont les secteurs protégés en écriture.
Pour protéger ou déprotéger des secteurs, il faut utiliser la commande `protect`. Il faut noter que les protections ne sont pas sauvegardées d'une remise sous tension à l'autre. Si vous souhaitez que les protections subsistent, il vous faudra modifier le code de u-boot et le re-flasher.
La Flash est effacée par la commande `erase`. **REMARQUE : il ne faut jamais utiliser la commande *erase all*(par mesure de sécurité elle est désactivée) car elle efface la totalité de deux bancs de Flash et donc les deux bootloaders. En cas d'erreur se précipiter pour couper l'alimentation de la carte.** *Si par malheur les deux bootloaders sont effacés alors il n'est plus possible de lancer le noyau du système d'exploitation et, plus grave, il n'est plus possible de reprogrammer les mémoires Flash depuis un des deux bootloaders. Dans ce cas il faut utiliser le port JTAG*[1] *pour reprogrammer ECMON sur un des deux bancs de Flash. Le brochage du port JTAG est donné au chapitre 3.11 du guide de l'utilisateur ACTIS[4]. Il faudra certainement*

[1] **JTAG :** Joint Test Action Group. Il s'agit d'un port dédié aux tests.

également contacter ACTIS pour obtenir le cable JTAG ainsi que logiciel pour utiliser ce port et pour obtenir une copie d'ECMON.
Enfin la copie de données se fait à l'aide de la commande `cp`.

ATTENTION : l'auto-reprogrammation ne fonctionne que très rarement. J'ai vérifié les algorithmes pour effacer et programmer les puces Flash et ils me semblent corrects. De plus je n'ai pas rencontré de problèmes lors de l'écriture d'autres secteurs que ceux de boot du banc de Flash numéro 2. De même la sauvegarde des variables d'environnement s'est toujours bien passée. Je soupçonne donc quelques adresses des puces de ne pas être très fonctionnelles. Mais peut-être est-ce moi qui me trompe. De toute façon en passant par ECMON pour programmer u-boot on ne rencontre jamais de problème.

Pour lancer le noyau

Entrez la commande `run bootcmd`

2.3 Le noyau Linux

Il faut d'abord aller dans les sources Linux. Dans le cas du CD fournit avec la carte PowerPC elles se trouvent dans le répertoire
`/opt/ecrin/crossdev/target/powerpc/src/linux`

A l'INRIA elles sont également disponibles dans le répertoire :
`$ROBOTIQUE/PowerPC/opt/ecrin/target/powerpc/src/linux`

Faire le ménage dans les sources :
`make mrproper`

Ensuite il faut configurer le noyau et les paramètres de compilation croisée avec la commande :
`make ARCH=ppc vsbc6862_config`

Si vous souhaitez voir et/ou changer cette configuration faire :
`make menuconfig` ou `make config`
Ne pas utiliser le commande *make xconfig*

Pour mettre à jour les dépendances, exécuter la commande :
`make dep`

Pour finir il faut compiler et créer l'image du noyau Linux. Pour ce faire utiliser une des commandes suivantes :

`make zImage`	contruit une image compressée du noyau
`make znetboot`	construit unr image compressée du noyau et l'écrit dans le répertoire `/tftpboot`

A présent vous disposez de l'image du noyau linux qui se trouve dans `/tftpboot` si vous avez utilisé la commande `make znetboot` ou alors dans le répertoire `arch/ppc/boot` des sources Linux si vous avez utilisé la commande `make zImage`.
Il est possible de booter cette image directement si vous utilisez ECMON comme boot loader, à condition de copier cette image dans `/tftpboot`
En revanche, si vous utilisez u-boot comme bootloader vous devez d'abord convertir le noyau pour qu'il soit exploitable par u-boot (voir le chapitre 2.2.3 pour les détails de la manipulation).

2.4 Le noyau RTAI

Deux cas peuvent se présenter : soit vous disposez déjà d'un noyau Linux patché avec RTAI soit il vous faut le faire vous-même.
Tout ce chapitre se trouve également dans un fichier README qui s'appelle `$ROBOTIQUE/PowerPC/RTAI/README.build_rtai`.

2.4.1 Vous avez déjà les sources prêtes

Dans ce cas, la procédure est exactement identique à celle pour Linux (voir chapitre 2.3). Il faut toutefois rajouter une étape supplémentaire : la compilation des modules RTAI.
Vous devez avoir deux répertoires comprenant des sources. Le premier contient les sources de linux patché pour fonctionner sous RTAI
(`$ROBOTIQUE/PowerPC/RTAI/actis-2.4.18`),
le second contient les sources de RTAI à proprement parlé
(`$ROBOTIQUE/PowerPC/RTAI/rtai-24.1.11`).
Il faut aller dans ce second répertoire. La première opération consiste à patcher les sources RTAI pour qu'elles fonctionnent avec la carte VSBC-6862. En effet, RTAI essaye d'utiliser un symbole qui n'est pas exporté ; j'ai donc tenté d'exporter ce symbole mais j'ai eu encore plus d'erreurs. Finalement, j'ai modifié les sources RTAI pour qu'elles utilisent un symbole exporté et qui fait exactement la même chose. J'ai donc crée un patch pour automatiser la modification. Les sources de RTAI sont dans le répertoire `$ROBOTIQUE/PowerPC/RTAI/rtai-2.4.11`.

Si vous avez mon patch pour RTAI

1. Allez dans le répertoire `arch/ppc` des sources RTAI
2. Exécutez la commande suivante :
   `patch < ../patch_rtai_vsbc6862`

Vous n'avez pas le patch

Vous devez alors faire les modifications à la main.

1. Allez dans le répertoire `arch/ppc` de RTAI
2. Editez le fichier `rtai.c`
3. Cherchez la ligne contenant la fonction `request_8xxirq`
4. Supprimez cet appel et les balises de compilation conditionnelle. Gardez l'appel à la fonction `request_irq`
5. Enregitrez les modifications

Configurez RTAI avec la commande :
`make menuconfig` (dans les sources RTAI)

Le programme vous demande le chemin des sources linux patchées, les lui donner.
Sortir du menu et enregistrer les modifications.
Ensuite exécutez les commandes suivantes :

```
make dep
make
make install INSTALL_MOD_PATH=/tftpboot/basefs-ppc
```

INSTALL_MOD_PATH est le chemin de la racine du système de fichiers monté par la carte au démarrage. Il lui sert à écrire les modules RTAI dans le `/lib/module` de la carte.

Installez les entrées dans le `/dev` de la carte :

```
make dev INSTALL_MOD_PATH=/tftpboot/basefs-ppc
```

MAIS il se peut que cette commande ne fonctionne pas du fait de problèmes de droits d'accès. J'ai donc récupéré la fin du makefile qui crée les entrées dans `/dev` et je l'ai recopié dans un script shell qui s'appelle `make_dev`.

1. Récupérez ce script dans `$ROBOTIQUE/PowerPC/RTAI` et copiez-le dans `/tftpboot/basefs-ppc/root` par exemple.
2. Bootez la carte PowerPC et connectez vous en utilisateur root
3. Lancez le script `make_dev`

RTAI doit être parfaitement fonctionnel à présent.

2.4.2 Linux n'est pas patché pour RTAI

Si vous avez uniquement les sources linux ACTIS alors il vous faut d'abord les préparer.

1. Copiez les sources linux de ACTIS dans le répertoire de votre choix.
2. Allez dans les sources copiées et exécutez les commandes suivantes :
   ```
   make mrproper
   make ARCH=ppc vsbc6862_config
   make dep
   ```
3. Appliquez le patch RTAI pour linux :
   ```
   patch -p1 <$ROBOTIQUE/PowerPC/RTAI/rtai-24.1.11/patches/-
   patch-linux-2.4.18-ppc-rthal5g
   ```
4. Exécutez :
   ```
   make dep
   make znetboot
   ```
   Remarque : les outils de compilation croisée doivent être dans votre PATH

Ensuite suivre les instructions du chapitre 2.4.1.

2.5 Compilation de busyBox

La carte PowerPC était livrée avec BusyBox 0.60.3. BusyBox[10] propose de nombreuses commandes shell simplifiées en un seul programme léger. Même si la version actuellement utilisée est encore la 0.60.3 j'ai compilé la version 0.60.5 pour la carte PowerPC, elle doit d'ailleurs être dans le répertoire `/bin` de la carte ACTIS.
Si vous souhaitez recompiler cette version 0.60.5 voici la marche à suivre :

1. Allez dans les sources de BusyBox 0.60.5. Elles sont dans le répertoire `$ROBOTIQUE/PowerPC/busybox/busybox-0.60.5`
2. Faites le ménage dans les sources avec la commande :
   ```
   $ make distclean
   ```
3. Compilez les sources avec la commande :
   ```
   $ make
   ```
4. Copiez l'exécutable `busybox` dans le répertoire `/tftpboot/basefs-ppc/bin`

2.6 Préparation de la carte PowerPC

Avant de commencer à utiliser les drivers et les programmes pour le robot bipède il faut configurer la carte processeur. Cette configuration est réalisée avec un script shell.
Ce script est dans le répertoire `$ROBOTIQUE/PowerPC/Bipede/scripts`.

Copiez le script `make_init_sequence` dans le répertoire `/root` de la carte, lancez la carte PowerPC et connectez vous en utilisateur root. Puis allez dans `/root` et lancez le script précédemment copié. Si vous exécutez la commande `ls` dans le répertoire `/root` vous devez voir un nouveau script dont le nom est `change_bip_mode`. Ce script permet de changer le mode de fonctionnement de la carte. Ainsi si vous tappez la commande suivante :

```
$ ./change_running_mode linux
```

Que vous vous déconnectez puis vous reconnectez, la carte doit fonctionner dans le mode linux. Pour connaitre tous les modes de fonctionnement disponibles tapez la commande :

```
$ ./change_running_mode
```

Vous devez également voir un script dont le nom est `get_bip_mode`. Ce script permet de savoir dans quel mode est la carte.

2.7 Compilation et installation des sources

2.7.1 Remarques générales

Tous les makefiles à part ceux pour VxWorks utilisent par défaut les outils de compilations dont le préfix est :
`$ROBOTIQUE/PowerPC/opt/ecrin/crossdev/host/powerpc/bin/powerpc-linux-`
Ils installent les librairies et les fichiers includes à partir de la racine `$HOME/Src` et l'installation des fichiers sur la carte PowerPC se fait à partir de la racine `/tftpboot/basefs-ppc/usr`. Il est possible de changer ces valeurs par défaut en modifiant les makefiles bien sûr mais également à l'aide de variables d'environnement. Ainsi la variable `CROSS_COMPILE_PPC` permet de changer le préfix des outils de compilation croisée. Pour la racine de l'installation il s'agit de la variable `INST_BASE` et pour l'installation sur la carte processeur la variable `BIP_BASE`.

2.7.2 Les drivers

Allez dans le répertoire `$ROBOTIQUE/Bipede/bipDriver/src` puis choisir le répertoire correspondant au système que vous souhaitez utiliser. Ensuite tapez les commandes :

```
$ make
$ make install
$ make install_bip
```

Si les drivers sont installés pour la première fois alors il faut créer les entrées dans `/dev` pour dialoguer avec les drivers sous Linux.
Pour celà il vous faut récupérer le script `make_dev` qui se trouve dans les sources Linux de `drvBip` et l'exécuter sur la carte :

```
$ cp $BIPDRV_DIR/src/linux/make_dev $PPC_BASEFS/root
```

Se connecter en root sur la carte

```
$ cd root
$ ./make_dev
```

2.7.3 Les librairies

Les librairies correspondent aux répertoires qui sont dans `$BIPEDE_DIR/utils`. Pour chacun de ces répertoires faire :

```
$ make -f Makefile.<nomDuSysteme>
$ make -f Makefile.<nomDuSysteme> install
$ make -f Makefile.<nomDuSysteme> install_bip
```

2.7.4 Les exécutables

Tous les exécutables liés aux drivers et aux librairies sont déjà copiés. Il ont été obtenus avec ORCCAD il suffir par conséquent de choisir le bon fichier de configuration lors de la phase de génération du code.

2.8 Utilisation des drivers

2.8.1 Sous Linux

Assurez vous que la carte PowerPC est bien en mode Linux : lorsque vous vous connectez à la carte le mode doit s'afficher. Il est également possible d'exécuter le script `get_bip_mode`. Si le mode n'est pas bon exécuter le script `change_bip_mode` avec le mode `linux` puis déconnectez-vous et reconnectez-vous.

Chargement des drivers

Il vous faut être utilisateur root et entrer la commande :

```
$ insdrv
```

Vous pouvez ensuite vous connecter sous un autre nom d'utilisateur pour lancer vos programmes Linux.

Lancer des programmes

Il vous suffit de taper le nom de programme pour que celui-ci s'exécute. Ou si ce n'est pas le cas mettre son chemin d'accès dans votre `PATH`.

Décharger les drivers

En utilisateur root tapez la commande :

```
$ rmdrv
```

2.8.2 Sous RTAI

Assurez-vous que la carte est en mode RTAI, voir chapitre 2.8.1.

Chargement des drivers

Passez en utilisateur root et tapez les commandes suivantes :

```
$ insdrv
$ insrt
```

Lancer les programmes

Se référer au répertoire qui contient les sources des programmes mais en général il vous faudra charger un ou des modules puis lancer un ou des programmes facultatifs.

Décharger les drivers

Connectez-vous en utilisateur root et tapez les commandes suivantes :

```
$ rmrt
$ rmdrv
```

Chapitre 3

Manuel du développeur

3.1 u-boot

u-boot est un bootloader libre distribué sur internet [6] avec une licence GPL. Il a été developpé à l'origine pour booter des cartes à base de processeurs PowerPC, mais à présent il supporte également d'autres architectures.
Dans notre cas nous avons besoin de la version pour PowerPC. Le support de nombreuses cartes PowerPC est fournit avec les sources, mais notre carte ACTIS ne fait pas partie des cartes supportées. J'ai donc du développer le support de notre carte ACTIS.
u-boot est plutôt bien fait et l'ajout du support d'une carte n'entraine la modification que de quelques fichiers. Je vais vous présenter les modifications que j'ai apporté.

3.1.1 Déclaration de la configuration de la carte

La première étape est de décrire la configuration de la carte. C'est à dire qu'il faut indiquer le nombre de bancs de Flash présents sur la carte, les adresses mémoires utilisées, etc...
Pour ce faire il faut rajouter une entrée dans le répertoire
`$ROBOTIQUE/PowerPC/u-boot/u-boot-0.4.0/include/configs`.
Pour notre carte j'ai crée le fichier `vsbc-6862.h`.
ATTENTION : les valeurs utilisées pour la configuration sont tirées du guide utilisateur fournit avec la carte ACTIS. Or certaines de ces valeurs se sont révélées incorrectes. Consulter le chapitre 3.1.3 pour voir les corrections apportées.
Initialement son contenu était la copie de celui d'une autre carte basée sur le processeur MPC8260. Les différentes valeurs des constantes ont ensuite été modifiées en fonction de la configuration de la carte ACTIS et des valeurs positionnées par ECMON.
Une fois ce fichier crée, j'ai crée un nouveau répertoire dans
`$ROBOTIQUE/PowerPC/u-boot/u-boot-0.4.0/board`.
Son nom est `vsbc-6862`. Ici aussi son contenu était initialement celui d'une autre carte basée sur le même processeur. Dans ce répertoire deux fichiers ont été modifiés : `vsbc-6862.c` et `flash.c`

`vsbc-6862.c`	contient la description de la configuration des ports d'entrée/sortie ainsi que le corps des fonctions spécifiques à la carte.
`flash.c`	contient les algorithmes pour lire, écrire et effacer la mémoire Flash.

Dans `vsbc-6862.c` j'ai modifié la configuration des ports d'entrée/sortie comme indiqué dans le `user's guide`[4] fourni avec la carte ACTIS. J'ai conservé inchangé la fonction d'initialisation du contrôleur SDRAM. Par contre j'ai modifié le contenu de la fonction `misc_init_r`. Cette fonction modifie la valeur de registres de la carte afin d'autoriser des transferts sur le bus

VME sur plus de 8 bits. Il s'agit des registres `VMAMA` et `VMAMB`. Enfin j'ai rajouté la fonction `convert_ubootbd_to_linuxbd`, nous verrons pourquoi plus tard.
Le fichier `flash.c` n'a pas été modifié si ce n'est par l'ajout de la reconnaissance de plus de types de mémoire flash. A cette occasion j'ai également modifié le fichier `$ROBOTIQUE/PowerPC/u-boot/u-boot-0.4.0/include/flash.h` : j'y ai rajouté le code de types de mémoire Flash non encore reconnus. En revanche je n'ai pas modifié les algorithmes de `flash.c`.
Enfin pour que tout ceci compile j'ai modifié le fichier `$ROBOTIQUE/PowerPC/u-boot/u-boot-0.4.0/Makefile` pour y rajouter la carte ACTIS.
Avec toutes ces modifications u-boot fonctionne mais Linux ne peut pas être lancé.

3.1.2 Support du Linux ACTIS

Lors de son lancement Linux prend en paramètre une structure qui contient divers paramètres tels que la fréquence d'horloge ou encore l'adresse Ethernet. Or la structure passée par u-boot n'est pas celle attendue par Linux. Heureusement la structure passée par u-boot contient toutes les informations nécessaires pour remplir la structure attendue par Linux. Dès lors deux solutions se présentaient à moi : soit modifier u-boot soit modifier le noyau Linux d'ACTIS. Il m'a semblé beaucoup plus prudent et judicieux de choisir le première solution.
Ainsi j'ai récupéré la structure attendue par Linux et j'ai copié sa définition dans le fichier `$UBOOT_SRC/include/configs/vsbc-6862.h`. J'ai ensuite crée la fonction `convert_ubootbd_to_linuxbd` dont j'ai parlé précédemment(3.1.1). Cette fonction se charge de recopier les champs nécessaires à Linux depuis la structure u-boot vers une structure adaptée à Linux. Il faut ensuite appeler cette fonction au bon moment. J'ai donc modifié le fichier `$UBOOT_SRC/common/cmd_bootm.c`. Ainsi juste avant d'appeler le noyau Linux j'appelle ma fonction de conversion.
Cette fois c'est bon, u-boot fonctionne et est capable de lancer le noyau Linux.

3.1.3 Les différences avec la documentation ACTIS

Lors du portage de u-boot j'ai découvert quelques erreurs dans le manuel utilisateur fourni par ACTIS[4]. Les corrections doivent figurer sur la version imprimée que j'ai utilisée mais je les décrirais ici en plus.

La mémoire Flash

Chapitre 4.2.4 : les colonnes 4 et 5 du tableau sur l'organisation des mémoires Flash en fonction de leur type sont inversées. Ainsi l'organisation du type 3 est en fait celle du type 4 et inversement.

La configuration du contrôleur SDRAM

Chapitre 4.3.2 : les valeurs de la figure 3.1 ont été utilisées pour configurer le registre `PSDMR` du contrôleur SDRAM, elles sont reprises de la configuration effectuée par ECMON(les lignes en gras contiennent des modifications). Pour en savoir plus sur la signification des champs consultez le guide utilisateur du processeur Motorola MPC8260[8] chapitre 11.3.3.

Configuration du bus pour les accès VME

Chapitres 4.6 et 4.7.3 : la configuration des chip select 4, 5, 6 et 7 est changée. Il faut adopter les configuratioms décrites par les figures 3.2 à 3.9. Pour en savoir plus sur la signification des champs consultez le guide utilisateur du processeur Motorola MPC8260[8] chapitres 11.3.1 et 11.3.2.

Bit	Champ	Valeur	Signification
0	PBI	1	oui
1	RFEN	1	oui
2-4	OP	000	normal
5-7	**SDAM**	**011**	**A16 est A5**
8-10	BSMA	010	non utilisé
11-13	A10	010	A8
14-16	**RFRC**	**100**	**6 tics**
17-19	PRETOACT	010	2 tics
20-22	ACTTORW	010	2 tics
23	BL	0	64 bits
24-25	LDOTOPRE	01	-1 tics
26-27	WRC	10	2 tics
28	EAMUX	0	pas de multiplexage
29	BUFCMD	0	non
30-31	CL	10	2 tics

FIG. 3.1 – Les valeurs corrigées de PSDMR utilisées dans u-boot

Bit	Champ	Valeur	Signification
0-16	BA	0xFA00	adresse de base
17-18	Reservé	00	—
19-20	PS	10	16 bits
21-22	DECC	00	Désactivé
23	WP	0	Désactivé
24-26	**MS**	**000**	**GPCM sur bux 60x**
27	EMEMC	0	Désactivé
28-29	ATOM	00	Désactivé
30	DR	0	non
31	V	1	Activé

FIG. 3.2 – Configuration modifiée du chip select 4 : BR4

Bit	Champ	Valeur	Signification
0-16	**AM**	**0xFFFF1**	**for 32ko**
17-18	**Réservé**	**00**	**—**
19	**BCTLD**	**0**	**Activé**
20	**CSTN**	**0**	**normal**
21-22	**ACS**	**00**	**0**
23	**Réservé**	**0**	**—**
24-27	**SCY**	**0000**	**0 tics**
28	**SETA**	**1**	**externe**
29	**TRLX**	**0**	**Désactivé**
30	**EHTR**	**0**	**Désactivé**
31	**Réservé**	**0**	**—**

FIG. 3.3 – Configuration modifiée du chip select 4 : OR4

Bit	Champ	Valeur	Signification
0-16	BA	0xC000	adresse de base
17-18	Reservé	00	—
19-20	PS	10	16 bits
21-22	DECC	00	Désactivé
23	WP	0	Désactivé
24-26	**MS**	**000**	**GPCM sur bux 60x**
27	EMEMC	0	Désactivé
28-29	ATOM	00	Désactivé
30	DR	0	non
31	V	1	Activé

FIG. 3.4 – Configuration modifiée du chip select 5 : BR5

Configuration des ports d'entrée/sortie

Chapitre 6.4 : La configuration des ports d'entrée/sortie est faite conformément au tableau disponible au chapitre 6.4 avec les modifications présentés par la figure 3.10.
Pour en savoir plus sur la signification des registres consultez le guide utilisateur du processeur Motorola MPC8260[8] chapitre 40.2.

3.2 Les drivers sous Linux

3.2.1 Modification des drivers VxWorks

Il existe deux différences majeures entre les drivers pour VxWorks et les drivers sous Linux-PowerPC.

Bit	Champ	Valeur	Signification
0-16	**AM**	**0xFE000**	**for 32Mo**
17-18	**Réservé**	**00**	**—**
19	**BCTLD**	**0**	**Activé**
20	**CSTN**	**0**	**normal**
21-22	**ACS**	**00**	**0**
23	**Réservé**	**0**	**—**
24-27	**SCY**	**0000**	**0 tics**
28	**SETA**	**1**	**externe**
29	**TRLX**	**0**	**Désactivé**
30	**EHTR**	**0**	**Désactivé**
31	**Réservé**	**0**	**—**

FIG. 3.5 – Configuration modifiée du chip select 5 : OR5

Bit	Champ	Valeur	Signification
0-16	BA	0xD000	adresse de base
17-18	Reservé	00	—
19-20	PS	10	16 bits
21-22	DECC	00	Désactivé
23	WP	0	Désactivé
24-26	**MS**	**000**	**GPCM sur bux 60x**
27	EMEMC	0	Désactivé
28-29	ATOM	00	Désactivé
30	DR	0	non
31	V	1	Activé

FIG. 3.6 – Configuration modifiée du chip select 6 : BR6

Bit	Champ	Valeur	Signification
0-16	**AM**	**0xFC000**	**for 64Mo**
17-18	**Réservé**	**00**	**—**
19	**BCTLD**	**0**	**Activé**
20	**CSTN**	**0**	**normal**
21-22	**ACS**	**00**	**0**
23	**Réservé**	**0**	**—**
24-27	**SCY**	**0000**	**0 tics**
28	**SETA**	**1**	**externe**
29	**TRLX**	**1**	**Activé**
30	**EHTR**	**0**	**Désactivé**
31	**Réservé**	**0**	**—**

FIG. 3.7 – Configuration modifiée du chip select 6 : OR6

Bit	Champ	Valeur	Signification
0-16	BA	0xE000	adresse de base
17-18	Reservé	00	—
19-20	PS	10	16 bits
21-22	DECC	00	Désactivé
23	WP	0	Désactivé
24-26	**MS**	**000**	**GPCM sur bux 60x**
27	EMEMC	0	Désactivé
28-29	ATOM	00	Désactivé
30	DR	0	non
31	V	1	Activé

FIG. 3.8 – Configuration modifiée du chip select 7 : BR7

Bit	Champ	Valeur	Signification
0-16	**AM**	**0xFC000**	**for 64Mo**
17-18	**Réservé**	**00**	**—**
19	**BCTLD**	**0**	**Activé**
20	**CSTN**	**0**	**normal**
21-22	**ACS**	**00**	**0**
23	**Réservé**	**0**	**—**
24-27	**SCY**	**0000**	**0 tics**
28	**SETA**	**1**	**externe**
29	**TRLX**	**1**	**Activé**
30	**EHTR**	**0**	**Désactivé**
31	**Réservé**	**0**	**—**

FIG. 3.9 – Configuration modifiée du chip select 7 : OR7

I/O	Fonction	PPARA	PDIRA	PODRA	PSORA
PA11	Rsrv2_IN	0	**0**	0	0
PA13	Rsrv4_MOD	0	**0**	0	0

FIG. 3.10 – Les modifications de la configuration des ports d'entrée/sortie

La première concerne l'accès au matériel. En effet là où VxWorks nous autorise à manipuler directement des pointeurs avec les vraies adresses physiques, Linux nous demande d'effectuer un "mapping" mémoire. Si cette transformation d'une adresse physique réelle en une adresse virtuelle n'est pas faite nous obtenons des erreurs de segmentation mémoire. Ainsi sous Linux avant de manipuler les registres des modules IP il est nécessaire d'appeler la fonction `ioremap` ou la fonction `ioremap_nocache` qui font exactement la même chose sous PowerPC. Cette fonction transforme une plage d'adresses réelles en une plage d'adresses virtuelles. Ensuite lorsque l'accès au matériel n'est plus nécessaire il faut appeler la fonction `iounmap` pour libérer les adresses virtuelles alouées.

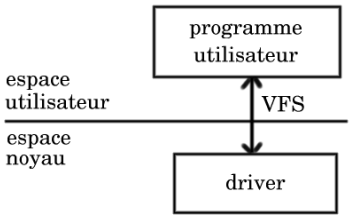

FIG. 3.11 – Organisation sous Linux

La deuxième différence concerne l'utilisation d'un driver. Sous VxWorks il existe un seul espace et toute fonction d'une librairie peut appeler une fonction d'une autre librairie. Sous Linux les choses sont différentes. En effet les drivers pour Linux ont été développés sous la forme de modules qui viennent se rajouter au noyau pour accroître ses fonctionnalités et, sous Linux, il existe une distinction entre l'espace noyau et l'espace utilisateur. Ainsi les fonctions d'un driver ne peuvent pas être appelées directement depuis un programme utilisateur. La communication se fait au travers du système de fichiers virtuel(VFS). Ainsi le driver est perçu comme un fichier. Par conséquent un programme utilisateur communique avec lui à l'aide des primitives de manipulation de fichiers telles que `open`, `read`, `ioctl`, ...
Je devais faire en sorte que le code des drivers puisse aussi bien être compilé pour VxWorks, pour Linux ou pour RTAI. En fait le code VxWorks est le cœur des drivers, tandis que des macros de compilation conditionnelle ajoutent les éléments spécifiques aux autres architectures. Ainsi sous Linux on doit ajouter un appel à la fonction `iomap` dans la fonction d'initialisation et un appel à la fonction `iounmap` dans le fonction de fermeture du driver. Il s'agit de deux modifications légères. En revanche l'introduction de l'interface VFS réclame plus de modifications. Heureusement il m'a été possible d'introduire toutes les fonctions relatives à cette interface à la fin des fichiers source et les fonctions de l'interface appelent directement les fonctions de l'interface VxWorks. Cette disposition permet de conserver un code lisible et aisément maintenable.

3.2.2 Les adaptateurs

Nous venons de voir (3.2.1) que la communication sous Linux et VxWorks est différente : sous VxWorks on appelle directement la fonction, sous Linux on utilise l'interface VFS. Par conséquent le programme utilisateur sous VxWorks et sous Linux n'est pas le même, toute sa communication avec les drivers est différente. Celà peut poser un problème puisqu'il faut adapter le code en utilisant de nombreuses macros de compilation conditionnelle. Certains bout de code sont dupliqués avec de légères modifications ce qui rend la maintenabilité du code plus complexe. J'ai donc cherché une solution qui puisse nous éviter au maximun de modifier le code déjà existant.
C'est ainsi que sont nés les adaptateurs. Il s'agit de bouts de code qui redonnent l'interface VxWorks aux drivers sous Linux. Ils dialoguent d'un côté avec le programme utilisateur et de

l'autre avec l'interface VFS.

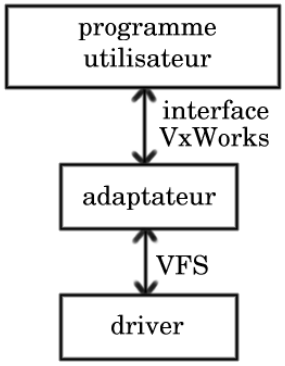

FIG. 3.12 – Place de l'adaptateur sous Linux

Ces adaptateurs effectuent l'opération inverse de celle qui a lieu dans les drivers : ils transforment les appels de fonctions VxWorks en appels de primitives de manipulation de fichiers. Grâce à ces adaptateurs il est inutile de modifier le code des librairies et des programmes utilisateur. Il suffit simplement lors de la compilation de rajouter les fichiers objets correspondants aux adaptateurs.

3.2.3 Un cas à part : le driver du module DACSU

Il s'est avéré lors des tests du driver pour le module IP DACSU de conversion numérique vers analogique, que le noyau Linux ACTIS ne veut pas d'opérations flottantes dans l'espace noyau. Ainsi lorsque l'on effectue des calculs sur des flottants un message s'affiche sur la console série pour nous prévenir, mais l'opération est tout de même effectuée. J'ai trouvé où ce message est affiché, il s'agit d'un petit bout d'assembleur. Je ne pouvais pas me permettre de conserver le fonctionnement du driver tel quel car les temps d'acquisition seraient dramatiques étant donné le débit de la liaison série utilisée pour afficher le message. Cette fois encore je voyais deux solutions : ou bien modifier le petit bout d'assembleur du noyau ou bien modifier mon driver. J'ai choisit le seconde solution car elle est moins risquée et au cas où nous changerions de version du noyau cette solution serait toujours valable.
Ainsi il s'agit de supprimer de l'espace noyau toute opération sur les nombres flottants. Ces opérations sont effectuées lors de la correction des consignes avec les gains et les offsets stockés sur le module IP. L'idée est donc de remonter ces données de calibration vers l'utilisateur pour que ce dernier effectue les corrections et retourne la valeur à écrire dans les registres de sortie. Cette situation explique pourquoi ce driver est plus modifié que les autres, surtout au niveau de la fonction `dacsuWrite`. Si l'adaptateur est utilisé tout ceci se fait de façon transparente pour l'utilisateur car l'interface proposée est encore identique à celle existante sous VxWorks.

3.2.4 Remarque concernant le module TIP501

Le driver pour le module IP TIP501 de conversion analogique vers numérique a été réécrit. Le but était simplement de lui donner le même type d'interface VxWorks que celle des modules GreenSpring. A l'origine ce driver utilisait une interface de type UNIX, c'est à dire qu'il proposait des fonctions telles que `open`, `read` ou encore `ioctl`.

3.3 Les drivers sous RTAI

3.3.1 Pour les modules IP

La version des drivers pour RTAI est très semblable à celle pour VxWorks. En effet les drivers seront utilisés par une tâche temps réelle qui sera par conséquent un module RTAI. Ce module RTAI pourra appeler directement les fonctions des modules driver. Ainsi l'interface VFS et les adaptateurs sont inutiles sous RTAI. Par contre il faut toujours transformer les adresses physiques réelles en adresses virtuelles.

3.3.2 Le cas DACSU

Le problème que nous avons évoqué dans le chapitre 3.2.3 n'est plus d'actualité. En effet lorsque l'on crée une tâche RTAI il faut indiquer si cette tâche utilise des calculs en nombres flottants. Il suffit par conséquent, lors de la création de la tâche RTAI qui utilise le driver, de la déclarer comme utilisatrice de calculs en nombres flottants.

3.3.3 Pour les librairies

Par contre cette fois les librairies `hardBip` et `drvBip` ont été modifiées.
`hardBip` a été transformé de façon à ce que l'on obtienne un module. Les modifications sont très localisées et peu nombreuses : il s'agit de déclarer qu'il s'agit d'un module et d'implémenter les fonctions `int init_module(void)` et `void cleanup_module(void)`.
`drvBip` pour sa part a été modifié pour pouvoir être compilée soit sous la forme d'un module soit sous la forme d'un fichier objet utilisable par un programme utilisateur. Les modifications sont identiques à celles apportées à `hardBip`.
Cependant un problème se pose à nous, il s'agit de la communication entre hardBip qui est un

module RTAI et drvBip lorsque ce dernier est utilisé au niveau utilisateur. Nous retrouvons un problème semblable à celui des drivers et des programmes utilisateurs sous Linux. Mais cette fois VFS n'est pas disponible, par contre RTAI propose plusieurs moyens de communication entre les modules RTAI et les processus Linux utilisateur. Parmis ceux-ci j'ai choisit d'utiliser les FIFOs car c'est un moyen très simple de communication. L'idée est de sérialiser les appels des fonctions de `hardBip` pour les faire passer par une FIFO puis de sérialiser les réponses de `hardBip` pour les faire passer par une deuxième FIFO. Les sérialisations et désérialisations sont effectuées par deux fichiers : `hardBip_serializer`, utilisé au niveau d'un processus Linux utilisateur et `hardBip_deserializer` un module inséré dans le noyau.

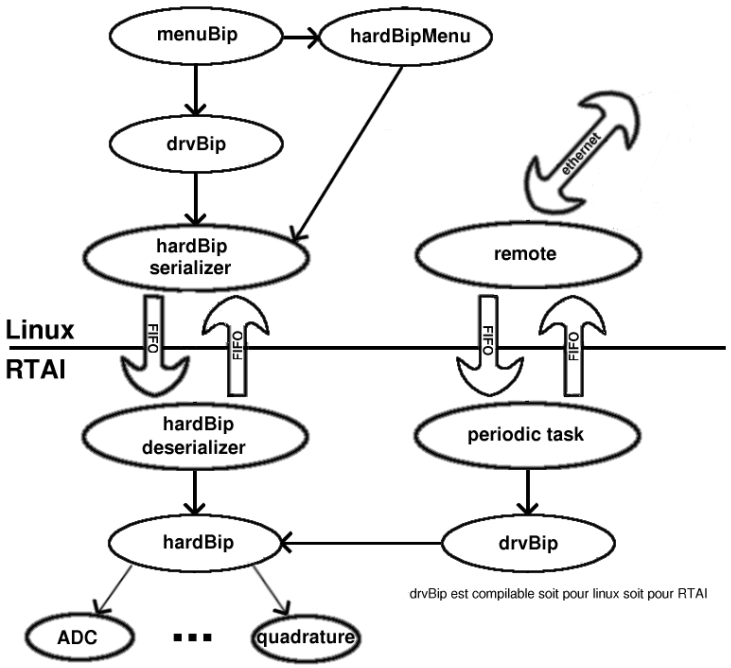

FIG. 3.13 – Organisation logicielle sous RTAI

3.4 Les librairies utilitaires

Il s'agit de librairies utilisées par les tâches ORCCAD. J'ai simplement crée des makefiles capables de créer des librairies adaptées à Linux et RTAI.

3.5 ORCCAD

ORCCAD[7] est un logiciel développé à l'INRIA Rhône-Alpes. Il s'agit d'un environnement graphique qui simplifie la conception de tâches. ORCCAD est composé :

- d'une interface graphique pour spécifier les tâches
- d'un générateur de code
- d'un runtime décliné sur toutes les plateformes supportées, Linux-PowerPC et RTAI-PowerPC entre-autres

3.6 Les tâches ORCCAD sous Linux

La première chose à faire est de vous procurer une version 3.1 d'ORCCAD capable de générer du code pour la cible LinuxPPC.

3.6.1 La tâche d'initialisation

C'est une tâche très simple. Comme la librairie `drvBip` est inchangée par rapport à la version VxWorks aucune modification de la tâche ORCCAD n'est nécessaire. En revanche il faut modifier les options de compilation. Pour ce faire il faut changer les paramètres dans le fenêtre ORCCAD "exécution". Il s'agit de mettre les bons chemins d'accès pour les fichiers include mais aussi d'indiquer les librairies utilisées par la tâche.

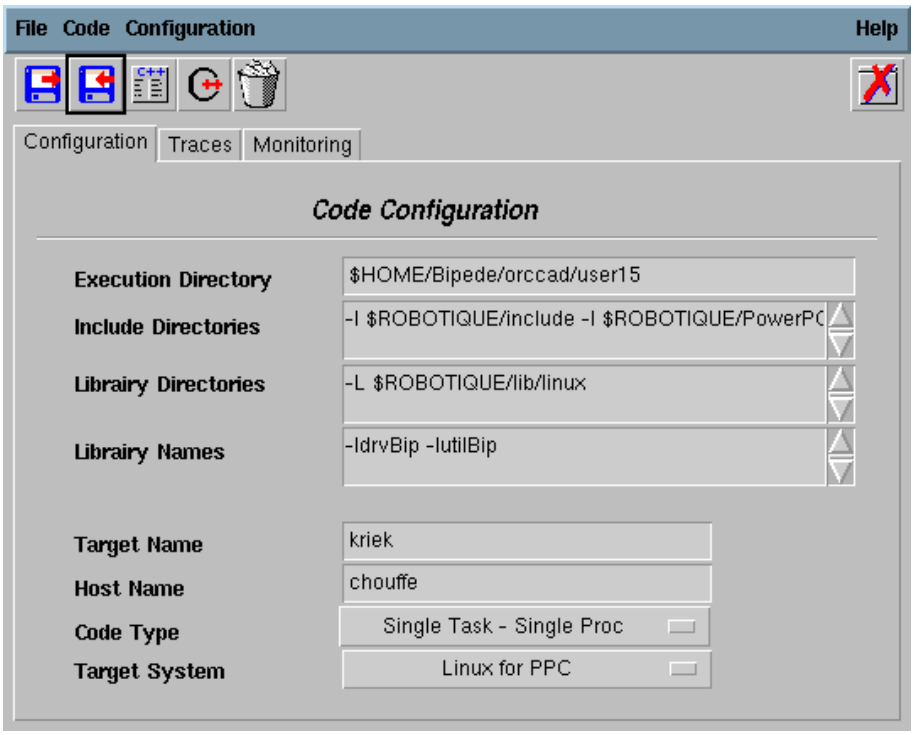

FIG. 3.14 – Un exemple de configuration du menu "exécution" d'ORCCAD pour Linux

3.6.2 La tâche de suivi de trajectoire

Cette tâche en revanche a été modifiée. En effet à l'origine elle utilisait une possibilté offerte par VxWorks mais pas par Linux.
Avant de jouer une trajectoire il faut initialiser le robot. Sous VxWorks cette initialisation se faisait en applelant une fonction chargée avec une librairie. Cette fonction modifiait les champs d'une structure partagée par la tâche ORCCAD également chargée sous VxWorks. Une fois la fonction d'initialisation terminée ont pouvait exécuter la tâche ORCCAD.
Or la transposition de cette approche nous donne deux programmes processus utilisateur sous Linux : un pour l'initialisation et un autre pour la tâche ORCCAD. Or sous Linux deux processus ne peuvent pas partager des données sauf au moyen de mécanismes particuliers. Ainsi la struture mise à jour par le programme d'initialisation n'est pas visible par la tâche ORCCAD.
Il a donc été nécessaire de modifier la tâche ORCCAD pour qu'en premier lieu elle fasse l'initialisation. Ainsi l'initialisation et la tâche de contrôle du robot ont lieu dans le même processus et la structure est la même pour les deux phases. Les modifications sont très légères car il a suffit de rajouter l'appel à la fonction qui lance l'initialisation.
Ensuite comme pour la tâche d'initialisation les plus grands changements ont lieu dans le menu "exécution" d'ORCCAD.

3.7 Les tâches ORCCAD sous RTAI

En premier lieu, vérifiez que votre version d'ORCCAD est capable de générer du code pour la cible RTAI-PPC.

3.7.1 La tâche de suivi de trajectoire

C'est la seule tâche ORCCAD portée sous RTAI. Ce portage a demandé plus de modifications que pour Linux. En effet la tâche de suivit de trajectoire fait appel à l'utilisateur lors de la phase d'initialisation du robot et lors du chargement des fichiers de trajectoire. De plus elle affiche des messages à destination de cet utilisateur. Malheureusement les fonctions si simples telles que `printf` ou `scanf` ne permettent pas de dialoguer avec l'utilisateur. J'ai donc crée un utilitaire supplémentaire utilisé par la tâche ORCCAD. Son nom est `InitManager`. Il se décompose en deux parties : un programme utilisateur exécuté par Linux et un module RTAI. Le programme Linux se charge du dialogue avec l'utilisateur et rempli les champs des structures locales correspondantes aux structures d'initialisation et de chargement de trajectoires. Une fois toutes les données nécessaires recupérées elles sont envoyées au module RTAI par l'intermédaire de FIFOs. Le module RTAI se charge alors de recopier ces données dans les structures utilisées par la tâche ORCCAD puis lance la tâche.
Par contre tous les messages à destination de l'utilisateur sont écrits à l'aide de la fonction `rt_printk`, c'est à dire qu'ils sont écrits dans le buffer circulaire du noyau. Ils sont donc visibles uniquement si la console série est connectée à la carte ACTIS ou si on utilise la commande `dmesg` dans une console virtuelle connectée à la carte.
Enfin comme pour le portage sous Linux il faut modifier les paramètres dans le menu "exécuter" d'ORCCAD.
Il faut noter que `drvBip` est utilisé dans ses deux formes : la forme Linux et la forme RTAI. En effet le programme utilisateur utilise `drvBip`, version Linux, lors de la phase d'initialisation du robot tandis que `drvBip` dans sa forme RTAI est utilisée par la suite par la tâche ORCCAD pour commander le robot.

3.8 Quelques suppléments pour RTAI

Dans le répertoire `/home/ciney/fuchet/Src/Bipede/utils` j'ai rajouté une librairie dont le nom est `UserDialog`.
A l'origine elle devait être utilisée pour porter la tâche ORCCAD de suivi de trajectoire vers RTAI(cf. 3.7.1). Cependant cette solution, si elle paraissait adaptée au dialogue homme/machine s'est révélée inadaptée au chargenent des trajectoires qui nécessite la lecture de fichiers. Ainsi cette librairie n'a finalement pas été utilisée. En revanche elle pourrait être utile dans d'autres cas c'est pourquoi je vais vous la présenter ici.
Elle utilise un programme utilisateur exécuté par Linux qui communique à l'aide de deux FIFOs avec un module RTAI. Le module RTAI propose des fonctions pour afficher des chaines de caractères, des entiers, pour demander à l'utilisateur de rentrer un nombre entier ou un flottant, etc... L'ensemble des fonctions proposées peut être trouvé dans le fichier `userDialog.h`. La méthode utilisée est semblable à celle employée pour la sérialisation/désérialisation lors du portage des drivers sous RTAI(cf. 3.3.3). Les appels aux fonctions du module RTAI sont sérialisés puis envoyés dans une FIFO. Ils sont reçus par le programme utilisateur qui les désérialise et les exécute. La valeur de retour, si elle existe, est alors sérialisée puis envoyée dans la deuxième FIFO. Enfin le module RTAI, qui était en attente, désérialise les données contenues dans cette deuxième FIFO et retourne le résultat au module ayant appelé la fonction de communication avec l'utilisateur.

3.9 Organisation de la carte PowerPC

3.9.1 Organisation des répertoires

Les librairies Linux et RTAI sont dans `/usr/lib/BIP/linux` et `/usr/lib/BIP/rtai`.

Les exécutables accessibles par le `PATH` de l'utilisateur sont dans `/usr/bin/BIP/linux` et dans `/usr/bin/BIP/rtai`.

Les modules Linux et RTAI sont dans `/usr/bin/BIP/linux/modules` et `/usr/bin/BIP/rtai/modules`.

Enfin les programmes de tests, qui ne sont pas dans le `PATH` de l'utilisateur sont dans `/usr/bin/BIP/linux/tests` et `/usr/bin/BIP/rtai/tests`.

```
/usr/+
     +lib/BIP/+
     |        +linux/
     |        +rtai/
     |
     +bin/BIP/+
              +linux/+
              |      +modules/
              |      +tests/
              |
              +rtai/+
                    +modules/
                    +tests/
```

FIG. 3.15 – Hiérarchie des répertoire sur la carte PowerPC

3.9.2 Gestion des modes Linux et RTAI

Quelle est l'idée ?

Linux et RTAI utilisent tous les deux les drivers des modules IP, mais ces drivers ne sont pas identiques, ils sont donc dans des répertoires différents. Sous Linux, comme sous RTAI, il existe des scripts `insdrv` qui insèrent ces drivers. L'idée est de cacher ces différences à l'utilisateur et de lui offrir une seule commande `insdrv` qui fonctionne pour Linux aussi bien que pour RTAI. Il s'agit de la même manière de positionner le `PATH` de l'utilisateur en fonction du mode de fonctionnement afin que les exécutables puissent êtres appelés depuis n'importe quel endroit de l'arborescence.

En pratique

Il s'agit de déterminer, lors de la connection de l'utilisateur, le mode de fonctionnement de la carte et de positionner les variables d'environnement en fonction du mode détecté.
Pour indiquer le mode de fonctionnement j'ai choisi de créer un fichier dont le nom est `bipLinux` si le mode de fonctionnement est Linux et `bipRTAI` si le mode est le mode RTAI. Ce fichier est crée à dans le répertoire `/etc` de la carte PowerPC.
Le fichier `/etc/profile` est modifié pour chercher ce fichier lors de la connection de l'utilisateur. Si le fichier `bipLinux` est trouvé l'environnement adapté au mode Linux est positionné par contre si le fichier `bipRTAI` est trouvé c'est l'environnement pour RTAI qui est choisit. Si ni l'un ni l'autre, ou au contraire les deux sont trouvés, alors l'environnement par défaut est conservé.
Pour rendre la modification du fichier `/etc/profile` plus facile j'ai crée un script dont le nom est `make_init_sequence`. Ce script se trouve dans le répertoire `$ROBOTIQUE/PowerPC/Bipede/scripts`. Si vous souhaitez modifier la phase d'initialisation lors de la connection vous pouvez modifier ce script.
Une petite remarque concernant ce script : lorsque vous l'exécutez il vérifie si le fichier `/etc/profile` contient déjà ou pas les modifications. Pour faire cette vérification il se fie à une clef. Si il trouve cette clef alors il en déduit que le fichier est déjà modifié, sinon il fait une copie de sauvegarde de l'ancien fichier `/etc/profile` et procède à la modification. A l'heure actuelle cette clef est *Clef : init_bip_env*. Il est donc conseillé de changer cette clef si vous faite une nouvelle version du script, sinon il ne mettra pas à jour le fichier `/etc/profile`.
Le script `make_init_sequence` crée également un script dont le nom est `change_running_mode`. Ce script permet de changer le mode de fonctionnement, et donc il ne fait que changer le nom du fichier à la racine.

```
export PATH=/bin:/usr/bin:/sbin:/usr/sbin
export LD_LIBRARY_PATH=/home/ppcuser/Jerome/lib/linux:\
                       home/ppcuser/Jerome/lib/rtai:\
                       /usr/lib/BIP/linux:/usr/lib/BIP/rtai
```

FIG. 3.16 – Le fichier /etc/profile avant d'être modifié par le script make_init_sequence

```
export PATH=/bin:/usr/bin:/sbin:/usr/sbin
export LD_LIBRARY_PATH=/home/ppcuser/Jerome/lib/linux:\
                       home/ppcuser/Jerome/lib/rtai:\
                       /usr/lib/BIP/linux:/usr/lib/BIP/rtai


# Clef: init_bip_env <------ La clef recherchee pour savoir si le fichier est a jour
# Regarde s'il doit demarrer en mode linux ou RTAI
if test -e /bipLinux && test -e /bipRTAI;
then
        echo Error cannot be in BIP linux mode and BIP RTAI mode at the same time;
elif test -e /bipLinux;
then
        echo Running in BIP linux mode

        export BIP_BIN=/usr/bin/BIP/linux
        export BIP_MOD=$BIP_BIN/modules
        export BIP_LIB=/usr/lib/BIP/linux
elif test -e /bipRTAI;
then
        echo Running in BIP RTAI mode

        export BIP_BIN=/usr/bin/BIP/rtai
```

FIG. 3.17 – Le fichier /etc/profile modifié par le script make_init_sequence

Bibliographie

[1] Site Internet d'ACTIS computers, constructeur de la carte processeur PowerPC
`http://actis-computer.com`

[2] Site Internet de RTAI
`http://www.aero.polimi.it/projects/rtai/`

[3] Le rapport de stage sur la mise à jour du contrôleur du robot bipède.
Mise à jour du contrôleur du robot bipède BIP2000
Rapport de stage, ESSI $3^{ème}$ année.
Rédigé par FUCHET Jérôme.

[4] VSBC-6862 User's Guide.
Manuel utilisateur fournit avec la carte ACTIS vsbc-6862.

[5] Le guide d'installation fourni avec la carte processeur ACTIS.
Linux support package for the vsbc-6862 - installation guide

[6] site Internet du bootloader u-boot :
`http://sourceforge.net/projects/u-boot/`

[7] Site Internet d'ORCCAD :
`http://www.inrialpes.fr/iramr/pub/Orccad/`

[8] Guide utilisateur du processeur PowerPC MPC8260.
Site Internet de motorola :
`http://e-www.motorola.com`
Le fichier s'appelle `MPC8260UM`.
A l'heure où j'écris ces lignes il est accessible à l'adresse :
`http://e-www.motorola.com/files/issue_files/cat_not_blade/MPC8260UM_D.pdf`

[9] Société ECRIN Systems, créatrice du boot loader ECMON.
Site Internet :
`http://www.ecrin.com`

[10] Site Internet de BusyBox :
`http://www.busybox.net`