Jelie : manuel de l’utilisateur

Julien Pilet et Stéphane Magnenat

Février 2003
$Revision : 1.18 $

Table des matières

Chapitre 1

Introduction

1.1 Qu'est-ce que Jelie ?

Jelie est un logiciel de pilotage d'interface JTAG pour les processeurs XScale d'Intel. Il permet de charger, d'exécuter et de déboguer des programmes. Il sert aussi à connecter GDB sur un PXA250 pour déboguer depuis un PC.

Sur une carte Armonie, Jelie permet également d'effacer et de modifier la mémoire flash. La nature évolutive de Jelie laisse ouverte la possibilité de l'étendre à d'autres processeurs.

1.2 Pré-requis

Jelie, pour fonctionner, a besoin d'un connecteur physique vers le processeur cible. Pour le moment, deux possibilités existent : un adaptateur sur port parallèle ou une carte EzUSB.

Jelie a été testé avec succès sous GNU/Linux sur x86 et sur Mac OS X sur PowerPC et fonctionne théoriquement sur tout système d'exploitation sur lequel la bibliothèque *libusb* a été portée.

Pour compiler Jelie, un compilateur croisé ARM est conseillé, car une partie du programme est embarquée sur le processeur cible. Le paquetage *binutils* de GNU fait très bien l'affaire. Le compilateur croisé *sdcc* doit être utilisé pour compiler le programme exécuté sur le microcontrolleur 8051 de la carte EzUSB. Si l'un ou l'autre de ces outils de développement croisé manque, les programmes embarqués ne seront pas recompilés et ceux distribués seront utilisés.

La bibliothèque *libusb* est nécessaire pour un support EzUSB.

1.3 Compilation et installation

Jelie utilise le système de gestion de projet GNU autoconf/automake[1]. Ce système adapte automatiquement les sources à la plateforme sur laquelle on les compile, sans avoir à les modifier. Il fonctionne notamment sous Unix (Linux), Mac OS X et Windows (grâce à Cygwin[2]).

Pour compiler Jelie à partir de la distribution officielle, il faut effectuer les étapes suivantes :

1. `gunzip < jelie-1.0.tar.gz | tar -xvf - && cd jelie-1.0` (décompresse Jelie et change de répertoire),
2. `./configure` (analyse le système et crée les fichiers `Makefile`),
3. `make` (compile le programme),
4. `make install` (installe le programme. Si l'installation est dans un répertoire système, l'installation doit être exécutée en temps que superutilisateur (*root*)).

Le fichier `INSTALL`, dans la distribution, explique ces étapes en détail.

[1] GNU Autoconf, Automake and Libtool `http://sources.redhat.com/autobook/`

[2] Cygwin est un environement Unix pour Windows. Cygwin est disponible à l'addresse `http://www.cygwin.com/`.

1.4 Principe de fonctionnement

Jelie a accès à la mémoire cache du processeur cible. L'architecture XScale permet en effet d'écrire et de valider des lignes de cache d'instructions. Une petite mémoire cache de 2 kilo-octets est d'ailleurs réservée pour le déboguage et ne peut être validée que par le port JTAG.

C'est dans cette mini-cache que Jelie va charger son gestionnaire de débogage (*debug handler*). Ce petit programme s'occupe, au démarrage, d'initialiser la mémoire vive et quelques périphériques comme le port série. Il est également appelé à chaque évènement de débogage (point d'arrêt, ou interruption externe par exemple).

Ce gestionnaire de débogage n'utilise aucune mémoire sur la machine cible. Il se sert de l'ordinateur hôte pour sauver et restaurer l'état du programme interrompu.

En communiquant avec sa partie embarquée, Jelie a, depuis l'ordinateur hôte, les possibilités suivantes :

1. lire et écrire des blocs de données à n'importe quelle adresse physique,
2. lire et modifier n'importe quel registre,
3. modifier les registres de débogage du co-processeur 15,
4. continuer l'exécution à une adresse donnée, avec un registre de statut donné.

Il y a deux moyens différents d'utiliser Jelie : une ligne de commande et GDB, via un port TCP/IP.

Chapitre 2

Utiliser Jelie

Avant de lancer Jelie, il faut s'assurer qu'il possède suffisamment de droits pour accéder à l'interface physique de connexion avec le processeur cible.

2.1 Exécuter Jelie

Jelie se lance depuis une ligne de commande. Certaines options permettent de modifier son comportement, avant même qu'il ne démarre. La ligne suivante aura pour effet de lancer Jelie avec connexion physique par carte EzUSB, en ouvrant le port 3141 pour GDB :

```
Jelie
```

2.1.1 L'option d'aide «-h»

Affiche un résumé des options disponibles et termine immédiatement.

2.1.2 L'option de débogage «-d»

Jelie lancé avec l'option «-d» affiche beaucoup plus d'information sur ce qui est en train de se passer. En particulier, toute la communication avec GDB est détaillée.

2.1.3 L'option port parallèle «-p»

Demande à Jelie d'utiliser le port parallèle plutôt que l'USB comme interface physique avec le processeur cible.

2.1.4 Ajout d'un chemin de recherche avec «-I»

A chaque fois que Jelie essaie d'ouvrir un fichier, il le cherchera dans une liste de chemins. Cette liste contient initialement le répertoire courant et le répertoire dans lequel Jelie a été compilé. Cette option, qui peut être mentionnée plus d'une fois, permet d'ajouter un chemin dans cette liste. Par exemple :

```
jelie -I /home/jpilet/jelie -I ../linux
```

Cette option est en majuscule pour être syntaxiquement compatible avec les options du compilateur GCC.

2.1.5 Choix d'un port TCP/IP avec «-t»

Jelie ouvre un port TCP/IP pour accepter une connexion provenant de GDB. L'option «-t» permet de choisir ce port. La ligne de commande suivante force Jelie à écouter le port 5000 :

```
jelie -t 5000
```

2.2 GDB/Insight et Jelie

Une fois que Jelie a démarré avec succès, on peut l'oublier et se concentrer sur GDB, dont la version avec interface graphique s'appelle Insight. Il suffit de le lancer, de choisir remote localhost :3141, de cocher la case 'load', et c'est parti.

2.3 La ligne de commande Jelie

Jelie peut afficher deux invitations à entrer une commande :

$$r >$$

ou

$$n >$$

Le *r* signifie que le gestionnaire de débogage est en train de s'exécuter, Jelie est donc prêt à lui faire faire son travail. Au contraire, un *n* indique que le gestionnaire de débogage n'est pas à l'écoute. Il faut peut être interrompre le programme qui tourne, redémarrer ou même recharger le gestionnaire de débogage avant de pouvoir à nouveau retrouver un fonctionnement complet.

Appuyer sur la flèche vers le haut rappelle la dernière commande entrée.

Appuyer sur tabulation après avoir écrit le début d'un nom de fichier le complétera si le début du nom est suffisant pour identifier un fichier existant.

La commande

```
info readline
```

affiche un guide d'édition de ligne de commande pour les programmes utilisant readline.

Si la bibliothèque libreadline n'est pas liée à Jelie, la ligne de commande a moins de fonctionnalités. En particulier, aucune invite n'est affichée, les noms de fichiers ne sont pas complétés par la touche tabulation, et la flèche vers le haut ne rappelle pas la dernière commande entrée.

Les commandes internes à Jelie ne prennent pas de «-» devant leurs options, contrairement aux arguments de Jelie lorsqu'il est lancé depuis la ligne de commande système.

2.3.1 La commande *help*

A tout moment, la commande *help* affiche la liste des commandes, avec une courte description. Si *help* est suivi d'une commande, par exemple

```
help help
```

une aide complète de la commande en question (*help* dans ce cas) est affichée.

2.3.2 La commande *quit*

La commande *quit* ne prend aucun argument. Son exécution force *Jelie* à terminer immédiatement.

Aucune commande n'est envoyée au processeur cible. Si un programme est en cours d'exécution, il n'est pas interrompu.

2.4 Initialisation

2.4.1 La commande *check*

Teste la connexion avec le processeur cible en lisant son numéro d'identification JTAG.

2.4.2 La commande *stop*

Arrête le processeur cible et le prépare à un chargement du cache d'instructions. Cette commande doit précéder la commande *loadic.*

Après le chargement de la mémoire cache, la commande *boot* peut redémarrer le processeur.

La commande *stop* ne prend aucun argument.

2.4.3 La commande *loadic*

```
loadic <fichier> <adresse> [mini]
```

Permet de charger un programme dans le cache d'instructions. Le fichier passé en paramètre est chargé dans le cache d'instructions, ou le mini-cache d'instructions si le mot-clé «mini» est spécifié comme troisième argument. L'adresse est donnée en hexadécimal, elle doit être multiple de 32 (0x20).

Le cache d'instructions est composé de lignes de 8 instructions. Si le fichier ne contient pas un nombre d'instructions multiple de 8, Jelie complétera avec des *nop*.

Plusieurs lignes de caches peuvent être chargées à des adresses non consécutives en appelant plusieurs fois la commande *loadic*.

Le mini-cache est de 2 kilo-octets et ne peut donc pas contenir plus de 512 instructions.

Le fichier donné en paramètre doit être un programme binaire petit boutien (*little endian*).

2.4.4 La commande *boot*

Permet de lancer l'exécution du processeur cible, à l'adresse 0x00000000. Si le cache d'instructions ou le mini-cache ont éte validés par la commande *loadic* à cette adresse, le programme exécuté sera celui contenu dans le cache. Sinon, la mémoire flash connectée sur le *chip select* 0 sera lue.

2.4.5 La commande *reset*

Effectue un *reset* du port JTAG sur le processeur cible. Cette commande ne sert qu'à faire des tests, elle est appelée automatiquement dans le cas d'une utilisation normale de *Jelie*.

2.4.6 La commande *reboot*

Redémarre le processeur cible, exactement comme si on appuyait physiquement sur le bouton *reset* de la carte Armonie.

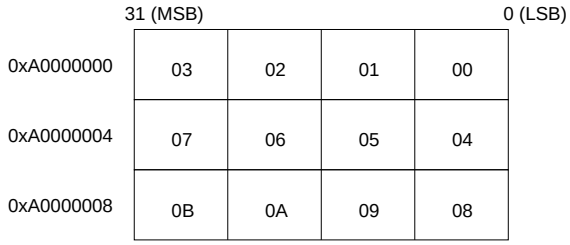
2.5 Accès mémoire

2.5.1 La commande *get*

```
get <adresse> [<nombre de mots à lire>]
```

Lit et affiche un ou plusieurs mots de 32 bits à partir d'une adresse donnée. Cette commande risque de causer une exception si l'adresse ne correspond à aucun périphérique.

Le résultat affiché est petit boutien (*little endian*). La figure suivante montre comment les octets individuels sont affichés. Le contenu de chaque case représente le numéro du byte.



Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.5.2 La commande *put*

```
put <adresse> <valeur> [h]
```

Écrit un mot de 16 ou 32 bits à une adresse donnée. L'adresse et la valeur sont à spécifier en hexadécimal. Si un «h» est passé en troisième argument, un mot de 16 bits est écrit, sinon, c'est un mot de 32 bits.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.5.3 La commande *load*

```
load <fichier> [<adresse> [h]]
```

Charge un fichier binaire à une adresse mémoire. Le fichier est lu puis envoyé par mots de 32 bits au processeur cible qui va écrire ces données à partir de l'adresse spécifiée.

Si un «h» est passé en troisième paramètre, l'écriture se fera sur 16 bits. Deux octets sur quatre du fichier source sont alors ignorés.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.6 Contrôle d'un programme

2.6.1 La commande *register*

```
register <numéro de registre> [<nouvelle valeur>]
```

Affiche ou modifie le contenu d'un registre.

Les registres conventionnels se comptent de 0 à 15. Le numéro 16 permet d'accéder au registre spécial *CPSR* du programme à exécuter. Le tableau suivant résume ceci :

N° du registre pour Jelie	Fonction du registre ARM
0	r0
1	r1
2	r2
3	r3
4	r4
5	r5
6	r6
7	r7
8	r8
9	r9
10	r10
11	r11, fp (*frame pointer*) in gcc
12	r12
13	r13, sp (*stack pointer*)
14	r14, lr (*link register*)
15	r15, pc (*program counter*)
16	CPSR (*current program status register*)

Modifier le registre 15 permet de faire un saut. Dans ce cas, le registre 15 contient l'adresse de la prochaine instruction à exécuter.

Les registres ne sont pas modifiés immédiatement, mais le seront dès que la commande *continue* dira au gestionnaire de débogage de continuer l'exécution normale du processeur cible.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5). S'il ne l'est pas, la lecture d'un registre affiche le contenu au moment où le gestionnaire de débogage a terminé son exécution pour la dernière fois et l'écriture n'a aucun effet.

2.6.2 La commande *continue*

Permet de démarrer ou de continuer l'exécution normale d'un programme.

Le processeur cible restaure tous les registres (voir la commande *register*) y compris le registre 15 contenant l'adresse de la prochaine instruction à exécuter et le registre de status *CPSR*.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.6.3 La commande *extbreak*

Interrompt l'exécution du programme en cours, sauvegarde les registres sur la machine hôte et entre dans le gestionnaire de débogage.

Cette commande n'a de sens que si un gestionnaire de débogage est chargé et si un programme cible est en cours d'exécution.

2.6.4 La commande *hwbreak*

```
hwbreak on|off [<address>] [0|1]
```

Contrôle les deux points d'arrêts matériel du processeur cible.

Un point d'arrêt permet d'interrompre un programme juste avant qu'il n'exécute l'instruction située à une adresse donnée.

Le premier paramètre est soit «on» soit «off», pour activer ou désactiver le point d'arrêt. Le deuxième est l'adresse, donnée en hexadécimal. Le troisième est soit 0 soit 1 : il permet de différencier les deux points d'arrêts matériels du XScale.

Par exemple :

```
hwbreak on a0000000 1
```

active le deuxième point d'arrêt à l'adresse 0xA0000000.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.6.5 La commande *wait*

La commande *wait* attend que le gestionaire de débogage soit prêt à exécuter une commande. Elle est utile principalement dans les scripts, pour s'assurer que le processeur soit prêt avant de continuer.

2.7 Opérations sur la mémoire flash

2.7.1 La commande *program*

```
program <filename> [address]
```

La commande *program* écrit dans la mémoire flash le fichier nommé *filename* à partir de l'adresse passée en paramètre. Si aucune adresse n'est donnée, l'écriture se fait à l'adresse 0.

Les secteurs concernés sont automatiquement effacés. Si le fichier passé en paramètre ne prend qu'une partie d'un secteur, l'autre partie est préservée (elle est copiée en mémoire vive pendant l'effacement puis réécrite automatiquement). Si l'adresse donnée ne correspond pas à la mémoire flash ou que le fichier est trop grand, rien ne se produit.

Cette commande ne peut être exécutée correctement que si le gestionnaire de débogage est prêt (voir 2.3 page 5).

2.8 Les scripts

Jelie permet d'exécuter des scripts qui regroupent des commandes. Ces scripts sont de simples fichiers textuels. Chaque ligne peut être vide ou contenir des commandes Jelie, un commentaire ou un appel à un autre script.

Toute ligne commençant par le caractère # est ignorée. Les lignes vides ou ne contenant que des espaces ou des tabulations sont également ignorées.

Pour exécuter une commande, l'interpréteur de Jelie cherche en premier dans la liste des commandes internes. S'il ne trouve pas, il cherche un fichier du nom de cette commande dans une liste de répertoires (voir 2.1.4 page 4). S'il trouve un tel fichier, il tentera d'en exécuter chaque ligne.

Les scripts peuvent prendre des paramètres. Avant l'exécution d'une ligne d'un script, toutes les séquences de la forme «\0», «\1», «\2», etc seront remplacées par le paramètre correspondant donné sur la commande d'appel au script. «\0» est toujours valide et correspond au nom du script appelé.

2.9 Changer de fréquence et passer en mode turbo

Ces commandes fonctionnent sur le PXA250 révision B. Rien ne permet de dire si elles marcheront sur d'autres révisions, et encore moins sur d'autres processeurs XScale. En effet, la documentation d'Intel ne correspond pas au comportement du PXA250 rev. B.

2.9.1 goturbo

Charge et exécute `turbo.bin`**.** Passe le processeur en mode turbo**.** Ce script modifie quelques octets à partir de l'adresse 0xa0000000.

2.9.2 loadlinux_turbo

Passe le processeur en mode turbo avant de charger Linux. Ce script modifie quelques octets à partir de l'adresse 0xa0000000.

Ce script a besoin d'un noyau et d'une image à charger. Il faut l'éditer pour mettre un chemin correct ou passer le répertoire en argument à Jelie avec l'option `-I`.

2.9.3 chspeed

Charge et exécute `fcs.bin`, ce qui change la vitesse du coeur à 200MHz et la vitesse du PXbus à 100MHz. Ce script modifie quelques octets à partir de l'adresse 0xa0000000.

2.9.4 loadlinux_fcs

Appel le script `chspeed` puis charge Linux. Ce script modifie quelques octets à partir de l'adresse 0xa0000000.

Ce script a besoin d'un noyau et d'une image à charger. Il faut l'éditer pour mettre un chemin correct ou passer le répertoire en argument à Jelie avec l'option `-I`.