

Développer un système embarqué avec la GNU toolchain

Jérôme POUILLER

Easter Eggs - Juin 2006

Table des matières

1	La <i>toolchain</i>	2
1.1	Utiliser une <i>toolchain</i> binaire précompilée	3
1.2	Compiler vous-même votre <i>toolchain</i>	3
1.3	Utilisation de scripts	3
1.3.1	CrossTool	3
1.3.2	BuildRoot	4
1.3.3	PtxDist	4
2	Le système de dépendances	4
2.1	Autoconf	4
2.2	Système maison	4
2.3	CMake	4
2.4	Automake	5
2.5	Compilation de binaire sur l'architecture hôte avec Automake	5
2.6	Les VPATH	5
2.7	Make distcheck	6
3	Initialiser votre cible	6
3.1	<code>crt0.s</code>	6
3.2	Les premières instructions	6
4	L'éditeur de lien	8
5	Utilisation d'une sonde BDI2000	9
5.1	Utilitaire de configuration	9
5.2	Microcode de la sonde	9
5.3	Configuration de l'interface réseau	9
5.4	Téléchargement par tftp	9
5.5	Structure du fichier de configuration	10
6	Gdb	11
6.1	Connexion par une sonde	11
6.2	Connexion par gdbserver	11
6.3	Interfaces pour gdb	11
	Références ¹²	

But du document

Ce document essaye d'aider le programmeur C classique à trouver son chemin pour programmer de systèmes embarqués.

Contexte

Ce document a été principalement rédigé par Jérôme Pouiller (jerome.pouiller@gmail.com), développeur système.

Il est surtout composé de retour d'expérience de mes divers projets au sein de la société Easter-Eggs¹, et en particulier d'un projet de système de reconnaissance de caractères. J'essaye de faire ici le point sur mes connaissances actuelles.

Remarque sur les anglicismes

Nous nous excusons auprès des puristes francophones. Ils trouveront dans ce document un certain nombre d'anglicismes. Ces anglicismes sont signalés par leur emphasement. Très souvent, nous ne connaissons pas la traduction française de ces mots et souvent le mot français est tellement peu utilisé qu'il n'apporte rien. Si quelqu'un trouve des anglicismes et connaît les traductions françaises, nous serions heureux de mettre à jour ce document.

Licence

Vous êtes libres :

- de reproduire, distribuer et communiquer ce document un tiers
- de modifier ce document
- d'utiliser ce document à des fins commerciales

Selon les conditions suivantes :

- Paternité : Vous devez citer le nom de l'auteur original.
- Partage des Conditions Initiales à l'Identique : Si vous modifiez, transformez ou adaptez ce document, vous n'avez le droit de distribuer le document qui en résulte que sous un contrat identique à celui-ci.

A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.

Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

La version complète de la licence peut être trouvée sur <http://creativecommons.org/licenses/by-sa/2.5/legalcode>

Configuration matérielle

Ce document est fortement lié au retour d'expérience d'Easter-Eggs sur un système de reconnaissance de caractères. Les principaux points techniques de ce système :

- La compilation est gérée par **Autotools**
- Le système est compilé à l'aide de **gcc** cross compilé pour ARM (diverses versions, principalement 3.4 et 4.0).
- Le système fonctionne sur une carte PC104 **Cogen CSB336**[4]
- La carte PC104 contient un processeur **MC9328MXL**[6]
- ... basé sur un cœur **ARM920T**[3]
- Le tout est connecté à un PC par une sonde **Abatron BDI2000**[1]
- La sonde contient un microcode² BdiGdb pour ARM7/9

1 La toolchain

Pour travailler sur une architecture distante, vous aurez sûrement besoin d'une *toolchain*. Une *toolchain* est un ensemble d'outils comprenant au moins un compilateur, un assembleur et un éditeur de liens³ exécutable sur une

¹<http://www.easter-eggs.com>

²firmware

³plus connu sous le nom de *linker*

architecture X produisant des binaires pour une architecture Y.

Vous avez trois solutions :

- Utiliser une *toolchain* binaire précompilée
- Utiliser certains scripts facilitant la compilation d'une *toolchain*
- Compiler vous-même votre *toolchain*

Remarque au sujet des normes C : On pense souvent que le C est un langage immuable. C'est faux. A chaque changement de versions majeures de gcc, la syntaxe du C change (je ne parle même pas des options et encore moins du passage sur un autre compilateur). Si devez compiler un code existant (tel que le noyau), renseignez-vous sur le compilateur utilisé et essayez d'utiliser le même. Sinon, vous aurez des problèmes de compilation à résoudre à la main (et parfois, ça n'est pas une simple question de syntaxe).

1.1 Utiliser une *toolchain* binaire précompilée

Si vous travaillez avec des compilateurs propriétaires, on vous fournira sûrement une *toolchain* précompilée. Vous pourrez aussi trouver quelques *toolchains* précompilées sur Internet. En particulier :

- CodeSourcery⁴ une excellente *toolchain* basé sur gcc.
- Eldk (Embedded Linux Development Kit)⁵ pour ARM, Mips et PowerPC. La distribution contient aussi une série d'outils précompilés pour l'architecture cible.
- Vous trouverez aussi un paquet Debian pour compiler pour palmOS.

L'installation d'une *toolchain* est souvent très simple (Une simple décompression d'archive suffit généralement). Néanmoins, il existe tellement de combinaisons compilateur/hôte/cible, que vous ne pourrez pas toujours trouver de *toolchain* précompilée adaptée à vos besoin.

Même si vous trouvez une version hôte/cible correspondant à vos besoins, vous aurez peut-être besoin d'avoir d'autre version de gcc pour compiler votre code⁶. Typiquement, toutes les versions du noyau Linux ne compilent pas avec toutes les versions de gcc

1.2 Compiler vous-même votre *toolchain*

Pour un maximum de souplesse, vous pouvez compiler vous même votre *toolchain*. Sachez que c'est un processus long et difficile. Il vous faudra parfois *patcher* le code pour qu'il fonctionne correctement.

Vous trouverez des informations plus détaillées sur <http://www.mega-tokyo.com/osfaq2/index.php/GCCCross-Compiler>

1.3 Utilisation de scripts

Heureusement, il existe un certains nombre de scripts vous permettant de fabriquer plus facilement un cross-compilateur.

1.3.1 CrossTool

CrossTool⁷ est ensemble de scripts pour générer des *toolchains* basées sur Gcc. C'est le script que je vous conseille d'utiliser. Il n'y pas de problème particulier à l'utilisation de CrossTool. Néanmoins, toutes les configurations ne fonctionnent pas. Vous aurez peut-être besoin de tester plusieurs configuration avant d'obtenir quelque chose fonctionnel. Je vous déconseille fortement d'essayer de *patcher* le code. Cela vous prendra souvent beaucoup de temps, et vous ne pourrez plus garantir le bon fonctionnement de votre *toolchain*. Vous pouvez visualiser les résultats des compilation sur <http://www.kegel.com/crosstool/crosstool-0.42/buildlogs/>. Normalement, si votre *toolchain* compile, il y a de très fortes chances pour qu'elle fonctionne correctement.

⁴http://www.codesourcery.com/gnu_toolchains/arm

⁵<http://www.denx.de/wiki/DULG/ELDK>

⁶surtout si vous essayez de compiler du code existant

⁷<http://www.kegel.com/crosstool/>

1.3.2 BuildRoot

BuildRoot⁸ va plus loin que CrossTool en tentant de compiler un système GNU/Linux complet. Il compile donc une *toolchain* gcc, puis les différents outils indispensables pour le bon fonctionnement d'un GNU/Linux (le tout basé sur la μ Libc) enfin, il installe le tout dans un répertoire sous la forme d'une arborescence complète. Vous n'avez plus qu'à transférer ce répertoire sur votre architecture cible.

Néanmoins, BuildRoot semble ne plus être maintenu depuis Février 2005. Je vous conseille donc de plutôt utiliser PtxDist.

1.3.3 PtxDist

PtxDist⁹ est l'outil qui se veut le plus abouti. Comme BuildRoot, il est capable de générer une arborescence GNU/Linux pour une architecture cible. Il utilise CrossTool afin de fabriquer sa *toolchain*. Vous pouvez de plus le configurer pour construire automatiquement un noyau ou bien simplement une *toolchain*. Enfin Il permet de plus de gérer facilement vos propres *patches*.

PtxDist associé à un serveur Subversion bien fait et quelques règles de *quality assurance* peut devenir un outil extrêmement puissant pour la fiabilité et la mise à jour automatique de votre système

2 Le système de dépendances

Pour compiler votre projet, vous aurez besoin d'un système de compilation possédant la notion d'architecture hôte et d'architecture cible. Vous avez en gros le choix entre utiliser Cmake, Automake et faire un système de compilation personnalisé. Si vous devez utiliser des Makefile lisez absolument "Recursive Make Considered Harmful" de Peter Miller[5], une référence.

2.1 Autoconf

Que vous utilisiez Automake ou votre propre système fait maison, rien ne vous dispense d'utiliser Autoconf. Autoconf :

- vous facilitera l'écriture de votre script configure (portabilité maximum)
- permettra de facilement gérer les options de compilation et principalement les options liées à la cross compilation
- vous permettra de gérer les VPATH de manière plus transparente.

2.2 Système maison

Si vous avez une bonne connaissance des systèmes de compilation, vous pouvez vous lancer dans un système personnalisé. Vous pouvez calquer votre système de compilation sur Kconfig (système de compilation du noyau Linux) qui est assez bien fait. Je lui reproche principalement d'être orienté arbre de compilation et non graphe de compilation à cause de sa structure récursive. Ensuite, la gestion des VPATH n'est pas aussi simple que dans Automake. Néanmoins, il possède une bonne gestion des différentes options de compilation et de la cross compilation. En particulier, il offre une interface graphique pour la configuration de votre système¹⁰. A partir de ces éléments, vous devriez pouvoir créer un système de compilation "from scratch" adaptés à vos besoins. Quoi qu'il en soit, préparez-vous à écrire un Makefile de plus de 2000 lignes !

2.3 CMake

Il y a encore quelques mois, CMake n'offrait pas de fonctionnalités suffisantes pour être utilisé pour la cross-compilation. Les choses ont bien évoluées depuis. Il semblerait CMake soit devenu le meilleur outils de compilation (en tous cas c'est ce que les caractéristiques technique montrent). A voir. . .

⁸<http://buildroot.uclibc.org/>

⁹http://www.pengutronix.de/software/ptxdist_en.html

¹⁰Remarquez que cette interface ne possède une utilité que si vous avez beaucoup d'options et que votre système doit être utilisé par un nombre assez important de développeurs

2.4 Automake

Vous pouvez aussi utiliser Automake. Automake est très puissant mais malheureusement, il souffre de quelques défauts sur les Makefile non-récurif (pas de variable indiquant le répertoire courant du fragment de Makefile) et au niveau de la cross-compilation (pas de gestion automatique de la variable `HOSTCC` comme avec KConfig).

L'utilisation d'Automake ne remet pas en cause l'article de Peter Miller[5]. L'écriture d'un Makefile non-récurif peut paraître un peu plus fastidieuse, mais elle vous évitera bien des soucis.

Si vous voulez un bel exemple d'utilisation de Autoconf/Automake, regardez les sources de ImageMagick

2.5 Compilation de binaire sur l'architecture hôte avec Automake

Vous aurez sûrement des programmes à compiler sur votre architecture hôte (génération de données, génération de programmes de tests, etc...). Malheureusement, il n'existe pas de cible du type `BUILD_noinst_PROGRAMS`¹¹ dans Automake. Il n'existe pas non plus d'options du type `ma_binaire_CC = gcc`¹². La meilleure méthode pour générer des données à l'aide de binaires compilés pour l'architecture hôte est sûrement :

```
data.h: $(genData_SOURCES)
        $(MAKE) $(AM_MAKEFLAGS) CC=$(BUILDCC) CCLD=$(BUILDCCLD) \
        CFLAGS="$$(BUILD CFLAGS)" genData
```

Les variables `$(BUILDCC)`, `$(BUILDCCLD)` et `$(BUILD CFLAGS)` doivent être gérées par Autoconf (ou votre propre script de configuration) :

```
AC_CANONICAL_HOST
[...]
if test x$cross_compiling = xyes; then
    BUILDCC=${BUILDCC:=${build_alias}-gcc}
else
    BUILDCC=${BUILDCC:=$CC}
fi
BUILDCCLD=${BUILDCCLD:=$BUILDCC}
```

Vous pouvez ajouter :

```
AC_ARG_VAR(BUILDCC,
    C compiler to build local tools (for cross compilation))
AC_ARG_VAR(BUILD CFLAGS,
    C compiler flags to build local tools (for cross compilation))
AC_ARG_VAR(BUILDCCLD,
    linker to build local tools (for cross compilation))
```

afin que d'officialiser l'utilisation des variables `BUILDCC`, `BUILD CFLAGS` et `BUILDCCLD` pour l'utilisateur.

Remarquez que nous n'avons ici que passé que les variables les plus utilisées de la compilation. Vous aurez peut-être besoin d'utiliser les variables `BUILDAS`, `BUILDASFLAGS`, etc...

2.6 Les VPATH

Utiliser `VPATH` consiste à compiler dans un répertoire différent des sources. Par exemple :

```
build\$\ ../src/configure
```

L'intérêt est d'avoir plusieurs configurations en même temps. Ainsi :

```
myproject\$\ mkdir final_arm debug_i386
myproject\$\ cd final_arm
myproject/final_arm\$\ ../configure CFLAGS="-O3 -DNDEBUG" \
    --host=arm-linux --build=i486-linux-gnu
myproject/final_arm\$\ cd ../debug_i386
myproject/debug_i386\$\ ../configure CFLAGS="-g-gdb3"
```

¹¹qui indiquerait que la cible doit être compilée pour la l'architecture locale

¹²D'ailleurs, si des développeurs Automake nous lisent, ces fonctionnalités seraient grandement appréciées

L'utilisation de VPATH est automatique dans Autoconf. Veuillez simplement à ce que votre code soit compilable de n'importe où. Utilisez pour cela les macros générées par Autoconf : @abs_top_srcdir@, @top_builddir@, etc. . . De même Automake gèrera automatiquement le changement de répertoire des sources. Il vous restera tout de même à faire attention à la gestion des scripts.

Si vous utilisez VPATH avec un système de Makefile non-récurrents¹³, n'oubliez pas que, sauf exception, toutes vos commandes sont exécutées à la racine de répertoire de compilation.

2.7 Make distcheck

Voici, à mon avis, le plus gros avantage d'Automake : la cible distcheck. Distcheck :

- récupère tous les fichiers déclarés dans Automake
- les compresse dans une archive
- décompresse les sources en lecture seule dans un nouveau répertoire
- compile dans un répertoire séparé des sources en utilisant VPATH
- installe le système dans une arborescence séparée des sources et de du répertoire de compilation
- exécute `make check` si la cible existe
- effectue un `make uninstall` puis, un `make distclean`
- vérifie que le répertoire, après le `make distclean` est dans le même état qu'à la décompression

Ainsi, en une seule commande, vous pouvez vérifier une grande partie de la cohérence de votre projet. Même si elle ne permet pas de voir tous les problèmes, le système est tout de même intéressant.

Parmi les options importantes, `DISTCHECK_CONFIGURE_FLAGS` permet de passer des options à configurer lors de la procédure.

3 Initialiser votre cible

Cette section décrit comment exécuter du code sur une architecture ne possédant pas d'OS.

3.1 crt0.s

Au détour d'un forum ou de vos recherches, vous entendrez parler de `crt0.s`. Ce fichier contient les premières instructions exécutées par un programme elf. Il initialise la pile, copie les données pré-initialisées dans une zone accessible en lecture/écriture et initialise le segment BSS (contenant les variables devant être initialisées à zéro) à zéro. Si vous utilisez un OS, ce fichier sera automatiquement utilisé par votre compilateur. Si vous travaillez sur une architecture sans OS, ce fichier n'est pas suffisant car il n'initialise pas le matériel (en particulier, il n'initialise pas la mémoire avant de l'utiliser et plantera certainement). Néanmoins, il pourra vous être très utile comme fichier d'exemple. En particulier, il utilise souvent des variables provenant de l'éditeur de liens.

3.2 Les premières instructions

Quelques conseils pour écrire la procédure de démarrage. Vous aurez besoin d'être proche de votre éditeur de lien et de votre processeur pour écrire cette procédure. Vous échapperez difficilement à quelques lignes d'assembleur.

Votre processeur démarre sûrement en plaçant PC à l'adresse `0x0`. Il est possible que le segment de mémoire `0x0` soit *mappé* matériellement (c'est à dire câblé sur votre carte) à deux adresses. Ce *mappage* permet très souvent d'utiliser une mémoire (typiquement, une mémoire flash) câblées plus haut dans la mémoire. Vous trouverez ces informations dans la documentation de votre processeur ou de votre carte¹⁴. Vous devez donc écrire un code (si possible relogeable en mémoire, pour éviter les problèmes liés au *mappage* de la mémoire) et le placer à l'adresse `0x0`. Déclarez un segment¹⁵. Par compatibilité avec les outils externes, tels que les binutils ou le débogueur, et les scripts ld appelez-le `.init` :

```
.section .init, "x"
.code 32 /* Always ARM mode after reset */
```

¹³Ce que je préconise

¹⁴Et si vous câblez votre propre carte, n'oubliez pas de mettre une mémoire sur l'adresse `0x0`

¹⁵En fait, vous pourriez utiliser le même segment que le reste de votre code pour placer votre code d'initialisation, mais très souvent, Vous aurez besoin que la procédure d'initialisation soit séparée du reste du code

Nous verrons plus tard comment demander à l'éditeur de lien de placer un segment à un endroit précis de la mémoire. Par compatibilité avec les outils externes, nous devons exporter le symbole `start`, `_start` ou `__program_start` suivant les outils. Le plus simple est de tout exporter :

```
.global __program_start
.global _start
.global start
```

Sur la plupart des processeurs, les premiers octets définissent la table des interruptions¹⁶. Par exemple, si l'interruption "IRQ" est déclenché, votre processeur sautera à l'adresse 0x18. A vous de vous débrouiller pour que l'interruption se déroule bien. Le premier élément du vecteur est généralement utilisé pour démarrer (et souvent, le premier élément se trouve à l'adresse 0x0, ce qui est bien pratique pour initialiser le processeur). D'une manière générale, les premières lignes de votre code ressembleront à :

```
.org    0x00
    ldr    pc, [pc, #24]
.org    0x04
    ldr    pc, [pc, #24]    /* Branch to undef_handler */
.org    0x08
    ldr    pc, [pc, #24]    /* Branch to swi_handler */
.org    0x0c
    ldr    pc, [pc, #24]    /* Branch to prefetch_handler */
.org    0x10
    ldr    pc, [pc, #24]    /* Branch to data_handler */
.org    0x14
    nop
.org    0x18
    ldr    pc, [pc, #24]    /* Branch to irq_handler */
.org    0x1c
    ldr    pc, [pc, #24]    /* Branch to irq_handler */
.org    0x20
.long   cstartup
.org    0x24
.long   undef_handler
.org    0x28
.long   swi_handler
.org    0x2c
.long   prefetch_handler
.org    0x30
.long   data_handler
.org    0x34
.long   undef_handler
.org    0x38
.long   irq_handler
.org    0x3c
.long   fiq_handler
```

Les directives `.org` permettent de placer le code suivant à une adresse précise relativement au début du segment. Vous remarquerez que la plupart de ses directives sont superflues, mais, elles permettent à l'assembleur de retourner une erreur en cas de mauvaise configuration. Dans notre exemple, le processeur commence à l'adresse 0x0 et saute directement sur `cstartup` ou nous pourrions commencer à réellement initialiser le processeur. En cas d'interruption telle que `undef_handler`, le processeur sautera en 0x4 qui l'emmènera dans la fonction `undef_handler` gérant réellement l'interruption.

Il ne nous reste maintenant plus qu'à déclarer une étiquette `cstartup` : et commencer l'initialisation du processeur et des périphériques de base. Vous devez maintenant vous reporter aux documentation des divers composants de votre système pour (dans l'ordre pour les cas les plus courants) :

¹⁶Vous pourrez trouver cette information dans la documentation de votre processeur

- Configurer l'horloge de votre processeur
- Configurer la RAM (en particulier, l'horloge, le rafraîchissement, etc. . .)
- effectuer le rôle de `crt0.s` :
 - Initialiser LES piles¹⁷
 - initialiser le segment BSS (c'est à dire les données initialisées avec 0).
 - initialiser les données accessible en lecture/écriture
 Pour toutes ces actions, vous aurez besoin de données issues de l'éditeur de lien. Notez que vous ne pouvez pas appeler de fonctions C de manière traditionnelle tant que que vous n'avez pas initialisé les piles.
- Activer le MMU¹⁹
- Activer les divers cache de mémoire. Généralement, il s'agit du cache de données et cache d'instructions. Notez que très souvent, à cause des algorithmes de *write-back* utilisés, le cache de données ne peut être activé que si le MMU est activé²⁰
- Configurer les différents périphériques : écran LCD, caméra, clavier, etc. . .

4 L'éditeur de lien

Si la cible sur laquelle vous compilez possède un OS et un système de fichiers, vous n'avez pas à toucher aux options de l'éditeur de lien. Sinon, vous aurez sûrement besoin de savoir comment fonctionne une binaire afin de configurer l'éditeur de lien.

L'éditeur de lien permet de spécifier à quel endroit de la mémoire chaque segment de programme doit être chargé. Il permet aussi de définir des emplacements pour la mémoire de travail : la pile, le tas, et éventuellement les entrées/sorties.

Je vous conseille de reprendre le script `ld` par défaut puis de le modifier pour vos besoin. Cela vous permettra d'être le plus compatible avec les outils extérieurs.

Quelques règles (assez logiques en fait) :

- Les segments demandant d'être permanent en mémoire doivent se trouver sur les plage en ROM (Flash). Il s'agit au minimum la procédure de démarrage de votre processeur (`.init`) et, à moins que vous ayez un périphérique de stockage, tout le code (`.text`, `.ctors`, `.dtors`, etc. . .) et les données (`.data`, `.rodata`).
- Les segments de mémoire de travail (`.bss`, `.stack`, . . .) doivent se trouver sur une plage de mémoire RAM
- Les segments de debug (`.debug*`) ne doivent pas être chargé dans la mémoire de l'architecture cible. Elle sont simplement utilisée par le débogueur.

La structure des scripts `ld` est majoritairement composé de directives

```
.nom : { *(.nom) }
```

ou

```
.nom adresse : { *(.nom) }
```

ou

```
__symbole__ = .;
```

Dans le premier cas, on demande à l'éditeur de lien de placer le segment suite au segment précédant automatiquement.

Dans le second cas, on demande à l'éditeur de lien de placer le segment à une adresse précise.

Enfin, la dernière syntaxe demande de définir un symbole qui aura pour valeur l'adresse actuelle (symbolisée par `."`). C'est ainsi que vous pouvez savoir où l'éditeur de lien a placé votre pile par exemple.

¹⁷Il est effectivement possible que le processeur (en particulier si il possède une MMU¹⁸) gère une pile contextuelle : pile système, pile superviseur, pile utilisatrice, etc. . .

¹⁹L'initialisation du MMU n'est pas triviale. Même si le principe est toujours de créer un table associative de blocs de mémoire, les structures changent beaucoup entre les processeurs. Très souvent la documentation du processeur donne un exemple de code

²⁰remarquez que le cache de donnée et le MMU peuvent induire des bugs difficile à trouver si ils sont mal configurés. Faites particulièrement attention à l'emplacement du vecteurs d'interruptions, aux plage d'adresse d'entrées/sorties et aux accès concurrent à la mémoire (cas courant : un DMA accédant à la mémoire par un processus asynchrone)

5 Utilisation d'une sonde BDI2000

La sonde BDI2000 de Abatron est une des sondes Jtags les plus populaires. Elle est compatible avec gdb et avec un peu de configuration, on peut l'utiliser dans les débogueur basé sur gdb tels qu'Emacs, kdbg, kdevelop, etc... Elle permet donc une grande souplesse d'utilisation. Néanmoins, sa conception commence à devenir obsolète et sa configuration peu paraître difficile.

Il existe bien d'autre sondes. Néanmoins, nous ne les avons pas testées et leur utilisation avec la *toolchain* GNU est souvent difficile (voir impossible).

5.1 Utilitaire de configuration

La BDI2000 utilise principalement une interface TCP/IP pour communiquer. La configuration de la sonde (et principalement de son interface réseau) se fait par le port série, à la manière d'un routeur. La sonde est livrée avec un utilitaire qui se compile sans problème.

Néanmoins, il semblerait que cet utilitaire soit bogué. Nous n'avons pas réussi à modifier correctement la sonde avec cet utilitaire. La sonde est aussi vendue avec une version Windows du programme de configuration. Ce programme fonctionne parfaitement sous wine. Grâce à la version Windows, nous avons réussi à correctement configurer la sonde.

Sur les version récent (9.4) de wine, il suffit de créer un lien symbolique de `/dev/ttyS?` et `/.wine/dosdevice/com1` puis de lancer l'utilitaire par `wine ./B20ARMGD.EXE`. La configuration réseau n'a besoin d'être faite qu'une fois (sauf bien évidemment en cas de changement de réseau)

5.2 Microcode de la sonde

La sonde BDI est une sonde Jtags générique. Seul le programme contenu sur la sonde permet de l'utiliser pour une architecture ou une interface particulière.

Votre sonde doit donc tout d'abord recevoir un microcode. Typiquement, vous aurez une ligne du genre :

```
$ ./bdisetup.d/bdisetup -u -p/dev/ttyS0 -aGDB -tARM
```

où

- La sonde Jtags connectée sur le port `/dev/ttyS0`²¹
- GDB est le nom de l'interface
- ARM est le type de processeur²²

5.3 Configuration de l'interface réseau

Typiquement, la ligne suivante permet de configurer l'interface réseau de la sonde.

```
$ ./bdisetup -c -p/dev/ttyS0 -i10.0.0.150 -m255.255.255.0
```

L'adresse IP est `10.0.0.150` et le masque de sous-réseau `255.255.255.0`.

Remarquez que la sonde permet aussi d'être configurée par Bootp

5.4 Téléchargement par tftp

Il faut maintenant donner à la sonde un fichier de configuration. La sonde est capable de récupérer ce fichier sur un serveur tftp. Je vous conseille l'utilisation de atftpd comme serveur tftp. Il est inclus dans la plupart des distributions et sa configuration est très simple²³. Cependant, n'oubliez pas que les serveurs tftp ne font généralement pas de translation de chemin. Si vous indiquez `/tftpboot` comme racine du serveur et que vous voulez récupérer le fichier `/tftpboot/boot`, il faudra spécifier `/tftpboot/boot` (et non `/boot` comme sur la plupart des serveurs http et ftp). Je vous conseille d'installer atftp (le client lié à atftpd) pour tester votre installation. Une fois votre serveur tftp en place, on spécifie l'adresse du serveur tftp et d'un fichier de configuration toujours par l'intermédiaire du port série :

```
$ ./bdisetup -c -p/dev/ttyS0 -h10.0.0.36 -f/tftpboot/config.cfg
```

²¹ Attention aux droit sur `/dev/ttyS0`

²² les anglophones diraient que GDB est le *frontend* et ARM le *backend*.

²³ triviale

5.5 Structure du fichier de configuration

Reportez vous à la documentation de la sonde BDI[1] pour une référence complète sur le fichier de configuration. Voici un exemple de configuration :

```
[INIT]
MMAP 0x00000000 0x000000FF
MMAP 0x00200000 0x00300000
MMAP 0x08000000 0x0C000000
MMAP 0x10000000 0x10800000

[TARGET]
; Cpu type
CPUTYPE ARM920T
; JTAG clock (0=Adaptive, 1=8MHz, 2=4MHz,...)
CLOCK 1
RESET HARD
; Memory model (LITTLE | BIG)
ENDIAN LITTLE
STARTUP RESET
;catch unhandled exceptions
;VECTOR CATCH
;the BDI working mode (LOADONLY | AGENT)
BDIMODE AGENT
;SOFT or HARD
BREAKMODE SOFT
;Workspace in target RAM for fast programming
WORKSPACE 0x08000000

[HOST]
IP 10.0.0.36
FILE /home/tftpboot/bin.elf
FORMAT ELF
;Load application MANUAL or AUTO after reset
LOAD AUTO

[FLASH]
;workspace in target RAM for fast programming
WORKSPACE 0x08000000
;Flash type (AM29F | AM29BX8 | AM29BX16 | I28BX8 |
CHIPTYPE STRATAX16
;The size of one flash chip in bytes (e.g. AM29F010
CHIPSIZE 0x00800000

;The width of the flash memory bus in bits (8 | 16 | 32)
BUSWIDTH 16
ERASE 0x10000000 ;erase sector 0 of flash in U12 (AM29F010)
ERASE 0x10004000 ;erase sector 1 of flash
ERASE 0x10008000 ;erase sector 2 of flash
ERASE 0x1000C000 ;erase sector 3 of flash
ERASE 0x10010000 ;erase sector 4 of flash
ERASE 0x10014000 ;erase sector 5 of flash
ERASE 0x10018000 ;erase sector 6 of flash
ERASE 0x1001C000 ;erase sector 7 of flash
ERASE 0x10020000 ;erase sector 4 of flash
```

Remarque : Le format de la binaire est fortement dépendante de l'éditeur de liens utilisé. En particulier, si vous souhaitez porter un nouveau linux, vous devrez utiliser `FORMAT BIN`

6 Gdb

Vous devrez utiliser une version de gdb compilée pour l'architecture cible. Très souvent, gdb a été avec le reste des outils de la *toolchain*.

6.1 Connexion par une sonde

Votre sonde est maintenant connectée au réseau et vous lui avez spécifié certains paramètres de base sur le type de système que vous souhaitez déboguer par l'intermédiaire du fichier de configuration. Il vous reste maintenant à vous connecter à votre sonde par gdb.

Gdb a besoin de connaître la binaire que vous utilisez afin de connaître les symboles de débogage et les sources de votre application. Il est important que cette binaire soit parfaitement synchronisée avec le contenu de la mémoire de votre système.

Ensuite, gdb est capable de se connecter à un serveur Gdb par l'intermédiaire de la commande `target remote hote :port`. Gdb utilisera ce serveur pour le débogage en lui-même.

Pour bien comprendre le fonctionnement de Gdb, il est important de rappeler que si vous avez correctement configuré votre éditeur de liens, les symboles de débogage ne sont pas chargés sur l'architecture cible. Seul gdb les lit. Ensuite, gdb est capable de convertir les adresses brutes de la sonde en nom de fonction/nom de fichier.

6.2 Connexion par gdbserver

Dans le cas où vous travaillerez sur un linux embarqué, vous pouvez utiliser la commande `gdbserver` sur votre cible. Naturellement, `gdbserver` doit être compilé pour l'architecture cible.

Il vous faudra ensuite configurer un peu gdb pour qu'il retrouve les informations de débogage.

Afin que gdb retrouve les bibliothèques chargées par votre programme. Ce préfixe sera ajouté à chaque demande de chargement d'une bibliothèque :

```
set solib-absolute-prefix {emplacement de votre arborescence cible}
```

Indiquer à gdb où se trouvent les informations de debug de la binaire :

```
file {votre binaire}
```

Se connecter :

```
target remote {hote}:{port}
```

6.3 Interfaces pour gdb

Vous n'avez peut-être pas la patience d'apprendre à utiliser gdb en ligne de commande. Il existe différentes interfaces :

- DDD
- kdevelop
- Emacs
- vim
- kdbg

Dans tous les cas, il faudrait modifier l'initialisation de gdb de manière à la faire correspondre avec votre environnement.

Conclusion

Nous avons passé en revue les difficultés que vous pourriez rencontrer dans le développement d'un système embarqué mais, il en existe d'autres. Si vous souhaitez faire des remarques ou des précisions sur le contenu de ce tutoriel, maillez-moi.

Références

- [1] A. AG, *bdiGDB, User Manual*, 1997. <http://www.abatron.ch/Files/ManGDBARM-2000C.pdf>.
- [2] D. M. TOM TROMÉY BEN PFAFF, *GNU Automake documentation*, GNU, 2002. http://www.gnu.org/software/automake/manual/html_mono/automake.htm.
- [3] ARM, *ARM920T, Technical Reference Manual*, 2000. http://www.arm.com/pdfs/DDI0151C_920T_TRM.pdf.
- [4] I. COGENT COMPUTER SYSTEMS, *Csb336 datasheet*, tech. rep., Cogent Computer Systems, Inc., 1130 Ten Rod Road, Suite A-201 North Kingstown, RI 02852, 2003. http://www.cogcomp.com/datasheets/Visio-CSB336_disti.pdf.
- [5] P. MILLER, *Recursive make considered harmful*, 1997. <http://www.pcug.org.au/millerp/rmch/recu-make-cons-harm.html>.
- [6] F. SEMICONDUCTOR, *MC9328MXL, Technical Data*, 2005. http://www.freescale.com/files/32bit/doc/data_sheet/MC9328MXL.pdf.