11/04

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Nationale Polytechnique

وزارة التعليـــــم العالـــــــي
و البحث العلمـــــي
المدرسة الوطنية المتعددة التقنيات

Département d'électronique

Projet de Fin d'études

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Thème:

Mise en oeuvre d'une plateforme de développement à base d'OpenRISC 1200

Etudié par :
Mme. Nezhate MAZOUZ.

Devant le jury composé de:

Dr. A.BELOUCHRANI	(maître de conférence)	Promoteur.
Mr. H.BOUSBIA-SALAH	(chargé de cours)	Président de jury.
Mr. Z.TERRA	(chargé de cours)	Examinateur.

Promotion juin 2004

E.N.P 10, Avenue Hassen Badi, B.P 182 El-Harrach, Alger, Algerie.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Table des matières

Résumé.....7
Abstract.....7
Résumé.....7
Mots clé.....7
Convention Typographiques.....8
Glossaire.....9
Introduction.....12
Chapitre 1: Architecture du microprocesseur OpenRISC.....15
1.1 Introduction.....15
1.2 Caractéristiques.....15
1.3 Détails de l'architecture.....16
1.3.1 Modes d'adressage.....16
1.3.2 Les opérandes.....17
1.3.2.1 Accès aligné et non aligné.....18
1.3.3 Les registres internes.....18
1.3.3.1 Registres à usage général (GPRs).....19
1.3.3.2 Registres à usage spécial (SPRs).....19
1.3.3.3 Registres vecteurs/virgule flottante (VFRs).....20
1.3.3.4 Registre superviseur (SR).....20
1.3.3.5 Les registres « Exception Program Counter » (EPCR0-EPCR15).....21
1.3.3.6 Les registres « Exception Effective Address » (EEAR0-EEAR15).....21
1.3.3.7 Les registres « Exception Supervision » (ESR0-ESR15).....22
1.3.3.8 Les registres: « next and previous programm counter » (NPC, PPC).....22
1.3.4 Le jeu d'instructions.....22
1.3.5 Le modèle des exceptions.....24
1.3.5.1 Introduction.....24
1.3.5.2 Les classes d'exceptions.....24
1.3.5.3 Traitement des exceptions.....25
1.3.5.4 Changement rapide du contexte « Fast context switching ».....27
1.3.5.5 Changement du contexte en mode superviseur.....27
1.3.5.6 Changement de contexte provoqué par une exception. 27
1.3.5.7 Accès aux registres d'autres contextes.....28
1.3.6 Organisation de la mémoire.....28
1.3.7 Gestion de la mémoire.....28
1.3.7.1 Utilité de la MMU.....28
1.3.8 La mémoire cache.....29
1.3.8.1 Les registres SPR de la cache.....30
1.3.9 Unité de débogage.....31
1.3.10 Les compteurs de performance.....31
1.3.11 Unité de gestion d'énergie.....31
1.3.12 Contrôleur d'interruptions programmable.....32
1.3.13 Tick Timer.....32
1.4 Implémentations existantes.....32

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

1.4.1 OpenRISC1200....................32
1.5 Évaluation du core....................35
Chapitre 2: Les outils de développement....................37
2.1 Introduction....................37
2.2 Les outils de développement GNU en résumé....................37
2.2.1 Les utilitaires....................38
2.3 Les librairies....................38
2.4 gcc, the GNU Compiler Collection....................38
2.5 cpp le préprocesseur GNU....................40
2.6 L'assembleur GNU as....................41
2.6.1 Format des fichiers objet....................41
2.6.2 La ligne de commande....................41
2.6.3 Les fichiers d'entrée....................42
2.6.4 les fichiers de sortie (objet)....................42
2.6.5 Les erreurs et les avertissements....................42
2.6.6 Les options de la ligne de commande....................43
2.6.7 La syntaxe assembleur....................43
2.6.7.1 Pré traitement....................43
2.6.7.2 Espaces....................44
2.6.7.3 Les commentaires....................44
2.6.7.4 Les expressions....................44
2.6.7.5 Les constantes....................44
2.6.8 Les symboles....................45
2.6.8.1 Les étiquettes....................45
2.6.8.2 Attribution des valeurs aux symboles....................45
2.6.8.3 Nom des symboles....................45
2.6.8.4 Nom des symboles locaux....................45
2.6.9 Les expressions....................45
2.6.9.1 Les expressions vides....................46
2.6.9.2 Les expressions entières....................46
2.6.10 Les directives assembleur....................46
2.6.10.1 Sections internes de l'assembleur....................47
2.7 L'éditeur de liens GNU ld....................47
2.7.1 Sections et relocalisation....................47
2.7.1.1 Utilisation des sections....................47
2.7.1.2 Les sections du linker ld....................48
2.7.1.3 Attributs des sections....................49
2.7.2 Les options de la ligne de commande....................50
2.7.3 Variables d'environnement....................51
2.7.4 Les script de l'éditeur des liens....................51
2.7.4.1 Le format du script linker....................51
2.7.4.2 Exemple de script....................51
2.7.4.3 Les commandes du script....................52
2.7.4.3.1 La commande sections....................52
2.7.4.3.2 La commande MEMORY....................53
2.7.4.4 Les expressions dans le script....................54
2.7.4.5 Le compteur de localisation....................54
2.7.4.6 Autres commandes....................54
2.8 Gcov, outil de test de couverture....................55
2.9 L'outil de debogage, gdb....................55
2.10 L'outil de recompilation, Make....................55

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

2.11 Binutils, les outils binaires....56
2.12 Le simulateur architectural....57
2.13 Installation des outils de développement....58
2.13.1 Introduction....58
2.13.2 Installation de gcc-2.95....58
2.13.3 Installation de binutils....59
2.13.4 Installation de gcc pour or1k....59
2.13.5 Installation de gdb 5.0....59
2.13.6 installation de or1ksim....60
2.13.7 installation d'uClinux....60
2.14 Installation de uClibc....61
2.15 Évaluation des outils....62
Chapitre 3: Conception d'un environnement de développement graphique pour OpenRISC....63
3.1 Introduction....63
3.2 Objectifs....63
3.3 Développement....64
3.3.1 Présentation de l'interface....64
Chapitre 4: Application : Filtre FIR....69
4.1 Conception du filtre et génération des signaux....69
4.2 Génèration des stimulus....74
4.3 Architecture choisie....77
4.3.1 Organisation mémoire....78
4.3.2 Configuration du simulateur....78
4.3.3 Le script Linker....78
4.3.4 Le programme de démarrage (boot)....78
4.4 Développement Filtre FIR....80
4.4.1 Les symboles utilisés....81
4.4.2 Utilisation des registres internes....81
4.4.3 Optimisations....81
4.4.4 Utilisation d'ORIDE....82
4.4.5 Compilation....82
4.4.6 Simulation et validation des résultats....83
4.4.7 Conclusions....84
Conclusions et perspectives....85
Conclusions....85
Perspectives....85
Références Bibliographiques....86
Annexes....87

Table des Illustrations

Figure 1.1: Adressage registre indirect avec déplacement....17
Figure 1.2: Adressage relatif au PC....17
Figure 1.3: Le jeu d'instruction....23
Figure 1.4: les valeurs des registres EEAR et EPCR après l'exception....26
Figure 1.5: le registre CXR....27
Figure 1.6: Transformation d'une adresse effective en adresse physique....29

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Figure 1.7: Architecture du processeur OR1200.........................33
Figure 1.8: Architecture du bloc CPU/DSP.........................34
Figure 2.1: Transformations du code source.........................40
Figure 3.1: Fenêtre principale d'ORIDE.........................65
Figure 3.2: Creation d'un nouveau projet.........................65
Figure 3.3: Fenêtre d'acceuil de l'assistant.........................66
Figure 3.4: Unités du CPU.........................66
Figure 3.5: Configuration des caches.........................67
Figure 3.6: Configuration du CPU.........................67
Figure 3.7: Périphériques à attacher.........................68
Figure 4.1: Définition du gabarit sous FDATool.........................70
Figure 4.2: Définition des paramètres du filtre.........................71
Figure 4.3: Définition des paramètres de quantification.........................71
Figure 4.4: les coefficients du filtre.........................72
Figure 4.5: exportation du filtre vers l'espace de travail MATLAB...73
Figure 4.6: fenêtre principale de genbench.........................75
Figure 4.7: nouveau projet.........................76
Figure 4.8: Définition des signaux d'entrée.........................76
Figure 4.9: État d'avancement de la génération des stimulus..........77
Figure 4.10: Architecture choisie.........................77
Figure 4.11: Algorithme du programme de boot.........................79
Figure 4.12: Changements du memory map par le programme de boot.........................80
Figure 4.13: visualisation des résultats.........................84

Table des Tables

Table 1.1: Les opérandes et leur taille.........................18
Table 1.2: Alignement des différents types d'opérandes.........................18
Table 1.3: Les groupes de SPRs.........................19
Table 1.4: Détails du registre SR.........................21
Table 1.5: Les classes d'exceptions.........................24
Table 1.6: Types d'exceptions et leurs causes, vecteur d'exceptions.... 25
Table 1.7: Registres du cache.........................30
Table 1.8: vecteur des exceptions.........................35
Table 4.1: Memory map.........................78

Résumé

Abstract

This work is an introduction to the software and hardware design using free tools and components. It's a starting point to OpenSource resources. We took as an example the development of softwares using the GNU toolchain on the OpenRISC1000 OpenSource core target. We also provide a set of tools to simplify the software development flow.

Résumé

Ce travail est une introduction aux logiciels et aux ressources matérielles (cores) libres. Il permet de se familiariser avec l'environnement de développement GNU sur une cible OpenRISC1000. Nous fournissons aussi un ensemble d'outils pour faciliter le développement d'application pour le processeur OpenRISC1000.

Mots clé

OpenSource, OpenRISC1000, ORIDE, GNU, Linux, uCLinux, GTK+, GNOME, RISC, core, FIR, genbench.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Convention Typographiques

Dans ce rapport, nous avons utilisés les conventions typographiques suivantes:

- Comande à taper dans la ligne de commande d'un terminal:

```
[shell]# commande
```

- Réponse affichée par un terminal:

```
1. line1
2. line2
3. ...
```

- Invite du simulateur architectural (+ commande à taper):

```
(sim)       commande
```

- Code source :

```
Code source d'un programme quelconque.
```

- Introduction d'un nouveau terme/abbréviation, se référer au renvoi ou au glossaire:

Mot

- Nom d'une commande :

commande

- Nom de fichier :

fichier.ext

- référence bibliographiques:

[REFXX]

- Le(s) bit(s) Y du registre X:

X[Y]

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Glossaire

Contexte: Le contexte du processeur est l'ensemble des informations qui déterminent sont état. Lorsque plusieurs programmes s'exécutent en temps partagé, il est necessaire d'avoir un mécanisme qui garantisse que chacun d'eux puisse retrouver le processeur dans l'état où il l'a laissé.

Cygwin: C'est le portage des outils GNU vers Windows.

Delay slot: lorsqu'une instruction de branchement est exécutée, l'instruction suivante sera automatiquement exécutée que le branchement ait lieu ou non. Ainsi, la sémantique du branchement change.

ELF: Format de fichiers objet.

GPR: Gneral Purpose Register, registre à usage général.

GTK: the Gimp Tool Kit c'est une librairie C permettant le développement des interfaces graphiques.

GNOME: c'est une librairie basée sur GTK, elle est utilisée pour le développement d'applications graphiques. Désigne aussi le bureau graphique GNOME sous linux qui est d'ailleurs basé sur la bibliothèque GNOME.

Harvard (architecture): Les données et les instructions sont sur deux bus différents.

LGPL: Library General Public License: c'est une licence spécifique aux librairies. Chaque librairie possédant cette licence garantie l'utilisation et la distribution gratuite des logiciels.

l.mtspr: instruction (assembleur) du processeur OpenRISC permettant d'écrire dans un registre à usage spécial. Mtspr= move to special purpose register

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

l.mfspr: instruction (assembleur) du processeur OpenRISC permettant de lire un registre à usage spécial. Mfspr= move from special purpose register.

Masque: En général, lorsqu'on veut activer l'effet de certains bits d'un registre, on lui applique un masque. Cela se fait en faisant un et binaire entre le masque et le registre en question. On place dans le masque un 1 pour les bits à conserver et un 0 pour les bits à ignorer.

PCI: Peripheral Component Interconnect : Branchement rapide entre un ordinateur et des cartes d'extension.

GNU: GNU's not UNIX. Projet d'un système d'exploitation OpenSource émulant UNIX.

Pile: Zone mémoire servant à la sauvegarde du contexte d'un processeur.

RISC: Reduced Instruction Set Computing/Computer: c'est un jeu d'instructions simplifié permettant des instructions s'exécutant en un seul cycle d'horloge.

SPR: Special Purpose Register : registre à usage spécial.

SMP: Symetric MutiProcessing: c'est une architecture multiprocesseur dont tous les processeurs ont le même droit pour accéder à une position mémoire donnée lorsqu'elle est partagée.

SMT: similtaneous MultiThreading: les systèmes d'exploitation ayant un seul processeur, ont la possibilité de traiter plusieurs processus et taches en même temps. Cette techniques est employée lorsqu'un processeur doit attendre des ressources externes pendant plusieurs milliers de cycles horloge. Durant ce temps, le processeur peut alors exécuter d'autres taches.

SR[SM]: Le bit SM indique si le processeur est en mode superviseur ou utilisateur.

SR[CID]: Donne le numéro du contexte actif.

SR[CE]: Désactive les registres ombres et CID. Les registres ombres sont des copies des GPRs.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

SR[SUMRA]: Autorise l'accès à certains SPRs en mode utilisateur.

Stanford (architecture): Les données et les instructions se partagent le même bus de données et d'adresses.

Superviseur (mode): Dans ce mode, le processeur a l'accès à l'ensemble des registres internes.

Timer: Composant générant des impulsions à des instants prédéfinis.

Thread: traitement, tache.

Utilisateur (mode): Dans ce mode, les programme ne peuvent pas accéder aux SPRs. Cela permet la protection du système car il n'y aura que le système d'exploitation qui pourra gérer le hardware.

Widget: Composant graphique de base entrant dans la composition d'interfaces graphique (équivalent à la notion de contrôle sous Windows)

Wishbone: Bus OpenSource pour l'interconnexion de cores dans un système sur puce (SoC: System on Chip)

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Introduction

De nos jours, l'informatique est devenue un outil indispensable à toute vie économique. Cela est la conséquence directe du développement de nouvelles techniques dans le domaine de l'informatique et de l'électronique.

Le marché de l'informatique s'est donc développé à une vitesse dépassant tout entendement. Mais le revers de la médaille est que les logiciels spécialisés deviennent de plus en plus chers, leur utilisation de plus en plus restrictive et soumise à des conditions limitant la liberté des utilisateurs en plus du fait que le code source n'est quasiment jamais fourni pour des raisons évidentes de « protection » posant de ce fait un problème supplémentaire quand à la maintenance des logiciels.

Face à cette situation, d'éminents informaticiens et chercheurs dirigés par Richard Stallman décidèrent au milieu des années 1980 de fonder la « Free Software Fondation » FSF dont le but est de développer et d'encourager les logiciels dits libres. Le terme libre ne signifie pas gratuit mais plutôt la liberté de redistribuer et surtout de disposer du code source de logiciel. Leur motivation première est que le produit d'un savoir ne devrait pas faire l'objet d'une commercialisation excessive car cela tuerait le savoir.

La FSF mis en place la licence « General Public Licence » *GPL* qui définit les droits juridiques des auteurs et des utilisateurs des logiciels, mais aussi des librairies (licence *LGPL*: L pour library) et la *GNU* Free Documentation License.

La FSF développa de nombreux outils qui devinrent des légendes dans le monde de l'informatique libre. Il s'agit principalement d'outils permettant le développement d'applications en vue de développer GNU (GNU's Not UNIX) qui est un projet de système d'exploitation libre émulant le fonctionnement d'UNIX. Les premiers outils développés furent Emacs (un éditeur de texte très riche), *GCC* (the GNU Compiler Collection) qui est un ensemble de compilateurs et différents outils de manipulation de fichiers binaires, d'assembleurs et autres.

De nombreuses institutions universitaires et personnalités éminentes du monde de la recherche scientifique ont pris part au mouvement insufflé par la FSF et ses fondateurs permettant ainsi un bond en avant dans la qualité et la stabilité des programmes développés par la communauté OpenSource. De nouvelles méthodes de développement de grands projets informatiques virent le jour car la particularité des projets issus de cette nouvelle philosophie et qu'ils sont caractérisés par le fait d'être développés par une grande communauté de développeurs disséminés aux quatre coins du monde, ce qui pose des problèmes autant de logistique et de coordination des efforts que de méthodologie de vérification.

Durant les années 90, le monde des logiciels OpenSource s'est considérablement développé. En effet, de nombreux projets comme Linux, GIMP (un logiciel de traitement d'images), APACHE (le serveur web le plus utilisé actuellement avec 60% de couverture), le langage PHP (langage de scripting très utilisé pour la création de sites web dynamiques), le navigateur web Mozilla sont devenus des logiciels fars et ont dépassés leurs équivalents commerciaux.

La philosophie de l'informatique libre gagna très rapidement le monde du développement électronique. Le projet le plus connus dans ce domaine est le projet OpenCores par l'intermédiaire duquel plusieurs cores OpenSource ont été développés.

Le code source de cores aussi variés que des unités arithmétiques et logiques, des codeurs/décodeurs audio et vidéo, des contrôleurs (USB, réseau, clavier, souris, VGA, LCD, mémoire SDRAM et Flash,), des FFTs, un bus pour systèmes sur puce (bus Wishbone), un pont PCI et même des processeurs.

L'avantage de ces cores est le fait qu'ils sont écrits en langages de description matérielle -Verilog en général car il existe un logiciel de synthèse, de compilation et de simulation de code Verilog, il s'agit d'ICARUS Verilog qui a atteint le niveau des logiciels propriétaires-. Ces cores sont donc synthétisables (et donc implémentables sur n'importe quelle technologie cible) et ce contrairement aux cores propriétaires qui sont fournis sous forme de netlist (liste d'interconnexions) propres à une technologie particulière (celle du fournisseur).

OpenRISC 1000 est l'un de ces cores. Il s'agit de processeurs OpenSource paramétrables à volonté (c'est à dire que le code source du processeur est paramétrable) et disponibles en version 32 ou 64 bits avec des extensions DSP, virgule flottante et de calcul vectoriel. Il couvre ainsi un large spectre d'applications.

L'un des avantages les plus importants de cette famille de processeurs est le fait qu'ils soient arrivés à maturité, les premières applications industrielles ont vu le jour au début 2004, par exemple: des décodeurs MPEG (vidéo compressée), un système de localisation par ondes TV, un système de reconnaissance vocale indépendante du locuteur. En effet un portage complet des outils de développement GNU et des systèmes d'exploitation (Linux uCLinux, eCos, RTEMS) a été effectué. Un simulateur architectural est aussi disponible, il est capable de simuler un système complet (Processeur et périphériques) à base d'un processeur OpenRISC et donc la validation d'une solution logicielle avant même la réalisation concrète de la carte. Il existe aussi une librairie C allégée (*uClibc*) qui est adaptée aux environnements réduits (systèmes embarqués).

L'objectif de ce projet de fin d'études est d'arriver à un point de maturité qui nous permettra de développer des solutions à base du processeur OpenRISC.

Pour atteindre cet objectif, nous nous sommes rendus compte qu'il est indispensable d'avoir une connaissance approfondie du système d'exploitation Linux [B01] et des outils de développement GNU, c'est pourquoi une grande partie de ce travail y est consacrée.

Il est indispensable aussi de se familiariser avec la famille OpenRISC qui est très riche en fonctionnalités plus ou moins avancées.

Une fois ces objectifs atteins et après avoir présenté quelques exemples d'applications, nous passerons au développement d'une application graphique qui aura pour but de simplifier le développement de logiciels ayant pour cible un processeur OpenRISC. Le but étant de démocratiser ce processeur en présentant une couche d'abstraction supplémentaire et ainsi d'éviter aux débutants de se heurter à la complexité des outils GNU, complexité qui constitue un facteur rebutant pour les débutants.

Dans ce travail, nous n'aborderons que très brièvement la partie développement Hardware, cela s'explique par le fait de la difficulté voire l'impossibilité de mettre en oeuvre tous les outils de développement (Hardware et Software) en quelque mois. Et surtout par le fait que la mise en oeuvre de toute solution commence d'abord en général par le développement de la partie Software puis par la partie Hardware.

Le rapport est ainsi organisé:

- Présentation du processeur OpenRISC.
- Présentation des outils de développement GNU.
- Développement d'un environnement de développement intégré pour le développement rapide d'applications à base d'OpenRISC.
- Implémentation d'un filtre FIR sur un système à base d'OpenRISC.
- Conclusions et perspectives.

Chapitre 1: Architecture du microprocesseur OpenRISC

1.1 Introduction

OpenRISC1000 [W05] est une famille de processeurs RISC haute performance, totalement gratuite et open source. Le code source est écrit en Verilog et soumis à la licence GPL. Les concepteurs d'OpenRISC1000 ont pris en compte les aspects de performances, de simplicité et 'économie d'énergie. La famille OpenRISC vise le marché des réseaux moyenne et haute performance et les systèmes embarqués.

La performance est assurée par une architecture 32/64 bits, des instructions pour le traitement vectoriel, DSP (cellule MAC, Multiply and ACumulate) et virgule flottante, gestion avancée de la mémoire virtuelle, la cohérence de la mémoire cache, support du *SMT* (Simultaneous MultiThreading) et *SMP* (Symmetric MultiProcessing) et le support du changement de *contexte* rapide.

Le concepteur est donc libre de choisir les fonctionnalités qui l'intéressent et ainsi de créer sa propre implémentation du processeur. Cela est grandement facilité par la possibilité d'étendre les instructions existantes, la possibilité de configurer le nombre de registres à usage général, la possibilité de configurer la mémoire cache, la gestion d'alimentation dynamique et enfin la possibilité de définir des instructions supplémentaires.

1.2 Caractéristiques

Parmi les caractéristiques de l'architecture OpenRISC1000[B04], on peut citer :

- Une architecture totalement ouverte (c'est à dire le code source est fourni) et gratuite.
- Un espace d'adresse logique linéaire sur 32 ou 64 bits et un espace d'adressage physique dépendant de l'implémentation.
- Un format d'instruction simple et uniforme caractérisant différents jeux d'instructions extensibles tels que:
  - ORBIS32/64 (OpenRISC Basic Instruction Set): Instructions sur 32 bits alignées sur 32 bits en mémoire et opérant sur des données de 32 et 64 bits.
  - ORVDX64 (OpenRISC Vector/DSP eXtension): Instructions sur 32 bits alignées sur 32 bits en mémoire et opérant sur 8, 16, 32 et 64 bits de données.
  - ORFPX32/64 (OpenRISC Floating-Point eXtension): Instructions sur 32 bits aligné sur 32 bits en mémoire et opérant sur 32 et 64 bits de données.
- Deux modes d'adressage mémoire où l'adresse effective est calculée par:
  - Addition d'un registre contenant l'opérande avec une valeur immédiate signée sur

16 bits.

- Addition d'un registre contenant l'opérande avec une valeur immédiate signée sur 16 bits suivi d'une mise à jour du registre contenant l'adresse avec la valeur de l'adresse effective déjà calculée.

- La plupart des instructions opèrent sur deux registres contenants les opérandes (ou un registre avec une constante) et le résultat est placé dans un troisième registre.
- Un fichier unique ou ombragé (c'est à dire plusieurs copies du fichier mais une seule copie visible à la fois) de 32 ou 16 registres à usage général. Ce qui facilite grandement le changement de contexte rapide.
- Branchement retardé afin de garder le pipeline aussi plein que possible.
- Séparation des espaces *cache* (cf 1.3.8 p29)/*MMU* (cf 1.3.7.1 p28) des données et les instructions (c'est à dire: Architecture *Harvard*) ou utilisation unifiée du cache/MMU (c'est à dire Architecture Stanford).
- La flexibilité de l'architecture offre la possibilité d'implémenter certaines fonctions soit sur matériel ou par assistance du logiciel.
- Support de différents types d'*exception* (cf 1.3.5.1 p24) simplifiant le modèle des exceptions.
- Support du changement rapide de contexte pour l'ensemble des registres, caches et MMUs.

1.3 Détails de l'architecture

1.3.1 Modes d'adressage

Lors d'un accès mémoire, branchement ou d'une opération de *fetch* (chargement de l'instruction), le processeur calcule l'adresse effective. Si la somme de l'adresse effective avec la taille de l'opérande dépasse la longueur maximale de l'espace d'adresse logique, la mémoire passe de l'adresse effective maximale à travers l'adresse effective 0.

Il existe deux modes d'adressage:

- Adressage registre indirect avec déplacement: Le contenu d'un GPR est additionné à un déplacement (sur 16 bit mais dont le bit signe est étendu à 32 bits).

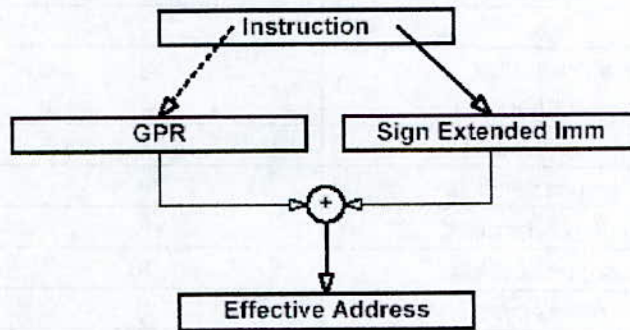


Figure 1.1: Adressage registre indirect avec déplacement

- Relatif au compteur programme (PC): La valeur immédiate (sur 24bits) est d'abord étendue sur 32bits (extension du bit signe) puis ajoutée à la valeur du PC.

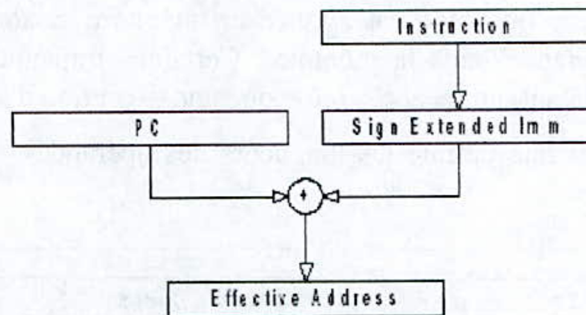


Figure 1.2: Adressage relatif au PC.

1.3.2 Les opérandes

Pour les opérandes, nous définirons la convention suivante:

TypeofData	LengthinBytes	LengthinBits
Byte	1	8
Halfword(or half) Singleword(or word)	2 4	16 32
Doubleword(or double)	8	64
Singleprecisionfloat	4	32
Doubleprecisionfloat	8	64
Quadprecisionfloat	16	128
Vectorofbytes	8	64
Vectorofhalfwords	8	64
Vectorofsinglewords	8	64
Vectorof singleprecisionfloats	8	64

Table 1.1: Les opérandes et leur taille

1.3.2.1 Accès aligné et non aligné

On dit qu'une opérande est alignée en mémoire si son adresse est un multiple de la longueur de l'opérande dans la mémoire. Certaines implémentations supportent l'accès non aligné, mais par défaut, un tel accès provoque une exception d'alignement.

La table suivante définie les longueurs des opérandes avec leur adresse dans le cas où elles sont alignées:

Operand	Length	addr[3:0] if aligned
Byte	8 bits	Xxxx
Halfword(or half)	2 bytes	Xxx0
Singleword(or word)	4 bytes	Xx00
Doubleword(or double)	8 bytes	X000
Singleprecisionfloat	4 bytes	Xx00
Doubleprecisionfloat	8 bytes	X000
Vectorofbytes	8 bytes	X000
Vectorofhalfwords	8 bytes	X000
Vectorofsinglewords	8 bytes	X000
Vectorofsingleprecisionfloats	8 bytes	X000

Table 1.2: Alignement des différents types d'opérandes.

1.3.3 Les registres internes

L'architecture OpenRISC1000 définit plusieurs types de registre: les registres à usage général et les registres à usage spécial (en *mode utilisateur*), les registres de contrôle et d'état du système (en *mode superviseur*) et les registres internes aux différentes unités (tels que: le *timer*, les caches, les MMU, l'unité de débogage).

L'architecture est Caractérisée par :

- 32 ou 16 registres à usage général à 32 ou 64 bits.
- 32 registres à 64 bits: vecteurs/flottants/DSP.
- Les autres registres sont à usage spécial, il sont définis séparément par l'unité correspondante et accessibles à travers les instructions: *l.mtspr/l.mfspr*.

Dans la partie qui suit nous allons détailler la description de certains registres essentiels.

1.3.3.1 Registres à usage général (GPRs)

Il existe 32 ou 16 (configuration à utiliser lors de l'implémentation sur FPGAs ou ASICS utilisés sur des systèmes embarqués dont les ressources sont limitées) registres à usage général, implémentés sur 32 bits et nommés R0-R31. Ils peuvent contenir des données ou des pointeurs.

Les *GPR*s (General Purpose Register) peuvent être des registres source et des registres destination selon l'instruction utilisée.

Le registre R0 est utilisé comme une constante nulle. Il ne doit en conséquence pas être utilisé en tant que registre destination.

Plusieurs jeux de registres à usage général peuvent être implémentés. Ces jeux de registres sont utilisés pour sauvegarder le contexte. Le passage d'un jeu à l'autre se fait dès qu'une nouvelle exception est atteinte. Le numéro du contexte actuel est placé dans *SR[CID]*.

l'initialisation des GPRs à zéro lors du démarrage (ou du reset) ne se fait pas de manière automatique, c'est au programme de boot (démarrage) de la faire.

1.3.3.2 Registres à usage spécial (SPRs)

Ces registres (*SPR*: Special Purpose Register) sont groupés en 32 groupes. Chaque groupe peut avoir un décodage d'adresse différent dépendant du nombre maximum théorique de registres contenus dans un groupe particulier.

Un groupe peut avoir des registres appartenant à différentes unités constituant le microprocesseur.

Certains registres sont accessibles uniquement en mode superviseur. Le bit *SR[SM]* est donc utilisé dans le décodage d'adresse des registres. Les instructions l.mtspr et l.mfspr sont utilisées pour la lecture et écriture dans ces registres.

GROUP#	UNIT DESCRIPTION
0	System Control and Status registers
1	Data MMU (in the case of a single unified MMU, groups 1 and 2 decode into a single set of registers)
2	Instruction MMU (in the case of a single unified MMU, groups 1 and 2 decode into a single set of registers)
3	Data Cache (in the case of a single unified cache, groups 3 and 4 decode into a single set of registers)
4	Instruction Cache (in the case of a single unified cache, groups 3 and 4 decode into a single set of registers)
5	MAC unit
6	Debug unit
7	Performance counters unit
8	Power Management
9	Programmable Interrupt Controller
10	Tick Timer
11-23	Reserved for future use
24-31	Custom units

Table 1.3: Les groupes de SPRs.

Un microprocesseur OpenRISC 1000 doit contenir au minimum les registres appartenant au groupe 0 (system control and status registers: pour configurer et vérifier l'état du processeur). Les autres groupes sont optionnels et leurs présence est conditionnée par la présence de l'unité correspondante.

Les bits [15-11] de l'adresse du SPR contiennent l'index du groupe et les bits [10-0] contiennent l'index du registres, le résultat constitue l'adresse d'un SPR.

Par exemple, pour calculer les adresses des registres suivants :

- Le registre *VR* (version register): l'index du groupe = 0, l'index du registre = 0, l'adresse est donc égale à 0x0000.
- Le registre CPUCFGR (CPU Configuration Register): l'index du groupe = 0, l'index du registre=2, l'adresse est donc égale à 0x0002.
- Le registre DMR2 (Debug Mode Register 2): l'index du groupe = 6, l'index du registre = 17, l'adresse est donc égale à 0x3011=(0b0011000000010001).

1.3.3.3 Registres vecteurs/virgule flottante (VFRs)

Ils sont 32 registres à 32 bits lorsqu'ils sont implémentés sur une architecture 32 bits mais dans ce cas, ils ne supportent pas les données de type: flottant double précision ou vecteurs. En version 64 bits par contre, ces types de données sont permis.

Les registre VFRs (Vector/Floating point Register) peuvent être des registres source ou

des destination, accessibles par des instructions de type : vecteur et à virgule flottante (ORVDX64,ORFPX32/64).

1.3.3.4 Registre superviseur (SR)

C'est un registre SPR à 32 bits, accessible uniquement en mode superviseur par les instructions l.mtspr et l.mfspr.

La valeur contenue dans ce registre définit l'état du processeur. La figure suivante décrit l'ensemble des bits de ce derniers :

Bit	31-28	27-17	16
Identifier	CID	Reserved	SUMRA
Reset	0	0	0
R/W	R/W	ReadOnly	R/W

Bit	15	14	13	12	11	10	9	8
Identifier	FO	EPH	DSX	OVE	OV	CY	F	CE
Reset	1	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	7	6	5	4	3	2	1	0
Identifier	LEE	IME	DME	ICE	DCE	IEE	TEE	SM
Reset	0	0	0	0	0	0	0	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 1.4: Détails du registre SR

1.3.3.5 Les registres « Exception Program Counter » (EPCR0-EPCR15)

Ce sont des registres SPR à 32 bits accessibles en mode superviseur par les instructions l.mtspr et l.mfspr.

Dans le mode utilisateur, l'accès à la lecture est permis si le bit *SR[SUMRA]* est activé.

Après une exception, le EPCR contiendra compteur programme (PC) de l'instruction qui a été interrompue par l'exception.

Si un seul des registres EPCR implémenté, il doit être sauvegardé par la routine d'exception et avant la réactivation des exceptions (par l'intermédiaire du SR).

1.3.3.6 Les registres « Exception Effective Address » (EEAR0-EEAR15)

Aussi, ce sont des registres SPR à 32 bits accessibles en mode superviseur par les instructions l.mtspr et l.mfspr.

Dans le mode utilisateur, l'accès à la lecture est permis si le bit SR[SUMRA] est activé.

Après une exception, le EEAR est mis à l'adresse effective (EA) générée par l'instruction qui a causée l'exception. Si seulement un seul EEAR est présent dans l'implémentation, il doit être sauvegardé par la routine d'exception et avant la réactivation de l'exception par le SR.

1.3.3.7 Les registres « Exception Supervision » (ESR0-ESR15)

Ce sont des registres SPR à 32 bits accessibles en mode superviseur par les instructions l.mtspr et l.mfspr.

Après une exception, le SR est copié dans un registre ESR. De la même manière, si seulement un seul ESR est présent dans l'implémentation, il doit être sauvegardé par la routine d'exception avant la réactivation de l'exception par le SR.

1.3.3.8 Les registres: « next and previous programm counter » (NPC, PPC)

Les registres compteur programme représentent l'adresse qui vient juste d'être exécutée et l'adresse de l'instruction suivante à exécuter.

Les registres: GPRs, NPC, PPC sont utilisés lors d'une opération de débogage par un debugger externe.

Pour avoir la valeur courante du compteur programme, il faut utiliser l'instruction l.jal et les instructions arithmétiques pour avoir la valeur du GPRs.

1.3.4 Le jeu d'instructions

Le jeu d'instructions de l'architecture OpenRisc 1000 est caractérisé par :

- Le format du jeu d'instructions qui est simple et uniforme. Il existe cinq sous ensembles d'instructions (cf Figure 1.3).
- ORBIS32/64 (OpenRISC Basic Instruction Set): Instructions de 32 bits, alignées sur 32 bits en mémoire et opérant sur 32 et 64 bits de données.
- ORVDX64 (OpenRISC Vector/DSP eXtension): Instructions sur 32 bits, alignées sur 32 bits en mémoire et opérant sur 8, 16, 32 et 64 bits de données.
- ORFPX32/64 (OpenRISC Floating-Point eXtension): Instructions sur 32 bits, alignées sur 32 bits en mémoire et opérant sur 32 et 64 bits de données.
- Codes opération réservés pour des instructions spéciales.

L'ensemble des instructions est divisé en deux classes. Seule l'implémentation de la classe de base est nécessaire.

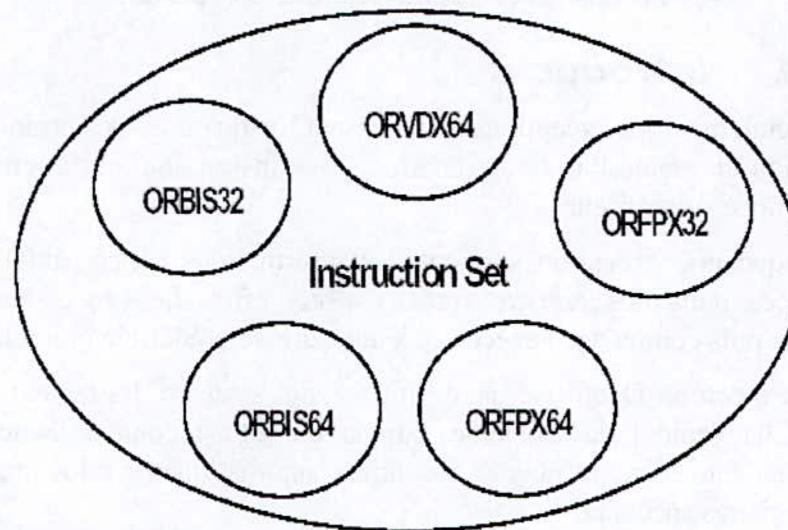


Figure 1.3: Le jeu d'instruction.

Les instructions appartiennent donc à l'un de ces sous-ensembles:

- ORBIS32:
  - Instructions à 32 bits opérants sur des entiers.
  - Instructions DSP de base (l.mac : pour effectuer un MAC).
  - Instructions de chargement/stockage sur 32 bits.
  - Instructions de gestion du flux (ex: branchement).
  - Instructions spéciales.
- ORFPX32:
  - Instructions à virgule flottante simple précision.
- ORFPX64:
  - Instructions à virgule flottante double précision.
  - Instructions de chargement/stockage sur 64 bits.
- ORVDX64:
  - Instructions de traitement vectoriel.
  - Instructions DSP.

Chaque sous-ensemble se divise en deux classes suivant l'importance de l'instruction. Les instructions de classe I doivent être toujours implémentées, mais celles de la classe II dépendent des fonctionnalités recherchées.

1.3.5 Le modèle des exceptions

1.3.5.1 Introduction

Généralement les exceptions apparaissent lorsqu'on a : des erreurs, un signal extérieur ou une situation anormale lors de l'exécution d'une instruction, elles permettent au processeur de passer en mode superviseur.

Lorsque une exception survient, les informations concernant l'état du processeur sont sauvegardées dans des registres prévus à cet effet. Le processeur passe alors en mode superviseur puis commence l'exécution à une adresse prédéfinie pour chaque exception.

L'architecture OpenRisc peut utiliser un système traitement rapide des exceptions, nommé: Changement de Contexte Rapide ou « Fast context switching ». Cette opération s'effectue en sauvegardant puis en restaurant automatiquement les registres à usage général et certains registres spéciaux.

Les exceptions doivent être exécutées dans leur ordre d'apparition, c'est pourquoi toutes les instructions se trouvant dans le pipeline et précédant l'instruction qui a causé l'exception doivent être exécutées avant de traiter l'exception en question.

1.3.5.2 Les classes d'exceptions

Toutes les exceptions peuvent être précises ou imprécises et synchrones ou asynchrones.

Lorsqu'une instruction provoque une exception, on dit qu'elle est synchrone. Elle est dite asynchrone dans le cas ou elle est provoquée par un évènement externe au processeur.

Le tableau suivant résume les classes d'exceptions :

Type	Exception
Asynchronous/nonmaskable	Bus Error, Reset
Asynchronous/maskable	External Interrupt, Tick Timer
Synchronous/precise	Instruction-caused exceptions excluding floating-point imprecise exceptions
Synchronous/imprecise	Instruction-caused floating-point imprecise exceptions

Table 1.5: Les classes d'exceptions.

Lorsqu'une exception survient, le contenu du PC est sauvegardé dans un registre EPCR et la nouvelle valeur du PC est donné (suivant le type d'exception) par le tableau suivant :

Exception Type	Vector Offset	Causal Conditions
Reset	0x100	Caused by software or hardware reset.
Bus Error	0x200	The causes are implementation-specific, but typically they are related to bus errors and attempts to access invalid physical address.
Data Page Fault	0x300	No matching PTE found in page tables or page protection violation for load/store operations.
Instruction Page Fault	0x400	No matching PTE found in page tables or page protection violation for instruction fetch.
Tick Timer	0x500	Tick timer interrupt asserted.
Alignment	0x600	Load/store access to naturally not aligned location.
Illegal Instruction	0x700	Illegal instruction in the instruction stream.
External Interrupt	0x800	External interrupt asserted.
D-TLB Miss	0x900	No matching entry in DTLB (DTLB miss).
I-TLB Miss	0xA00	No matching entry in ITLB (ITLB miss).
Range	0xB00	If programmed in the SR, the setting of certain flags, like SR[OV], causes a range exception. On OpenRISC implementations with less than 32 GPRs when accessing unimplemented architectural GPRs. On all implementations if SR[CID] had to go out of range in order to process next exception.
System Call	0xC00	System call initiated by software.
Reserved	0xD00	Reserved for future use.
Trap	0xE00	Caused by the l.trap instruction or by debug unit.
Reserved	0xF00	Reserved for future use.
Reserved	0x1000 – 0x1800	Reserved for implementation-specific exceptions.
Reserved	0x1900 – 0x1F00	Reserved for custom exceptions.

Table 1.6: Types d'exceptions et leurs causes, vecteur d'exceptions

1.3.5.3 Traitement des exceptions

Lorsqu'une exception survient, le contenu du PC est sauvegardé dans un registre EPCR sauf si l'instruction courante se trouve dans le *delay slot*. si le PC pointe une instruction delay slot, PC-4 est sauvegardé dans le EPCR, le bit SR[DSX] est mis à 1 et le SR est sauvegardé dans le registre ESR (Exception Supervision Register) (cf 1.3.3.7 p22).

Par ailleurs, le registre EEAR (cf 1.3.3.6 p21) courant contient l'adresse effective en question si une de ces exceptions parvient: bus error, IMMU page fault, DMMU page fault, alignment, I-TLB miss, D-TLB miss.

Le tableau suivant définit les valeurs des registres EEAR et EPCR après une exception:

Exception	Priority	EPCR (no delay slot)	EPCR (delay slot)	EEAR
Reset	1	-	-	-
Bus Error	4 (insn) 9 (data)	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Load/ store/fetch virtual EA
Data Page Fault	8	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Load/store virtual EA
Instruction Page Fault	3	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Instruction fetch virtual EA
Tick Timer	11	Address of next not executed instruction	Address of just executed jump instruction	-
Alignment	6	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Load/store virtual EA
Illegal Instruction	5	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Instruction fetch virtual EA
External Interrupt	11	Address of next not executed instruction	Address of just executed jump instruction	-
D-TLB Miss	7	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Load/store virtual EA
I-TLB Miss	2	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	Instruction fetch virtual EA
Range	10	Address of instruction that caused exception	Address of jump instruction before the instruction that caused exception	-
System Call	7	Address of next not executed instruction	Address of just executed jump instruction	-
Trap	7	Address of next not executed instruction	Address of just executed jump instruction	-

Figure 1.4: les valeurs des registres EEAR et EPCR après l'exception.

Si le FCS (Fast Context Switching) (cf 1.3.5.4 p27) est utilisé, la valeur SR[CID] est incrémenté à chaque nouvelle exception faisant passer le processeur à un nouveau contexte.

Si SR[CID] provoque un dépassement lors de l'exception courante, une exception de rang survient. Cependant, si le bit ***SR[CE]*** est désactivé, le FCS est désactivé et dans ce cas tous les registres qui vont être utilisés pour l'exécution de l'exception doivent être sauvegardés (par l'utilisateur et donc le programme de gestion des exceptions) en premier lieu.

Une fois que l'exception est exécutée, la restauration de l'état du processeur (SR et PC) se fait à l'aide de l'instruction l.rfe (return from exception). Si le bit SR[CE] est activé, SR[CID] sera automatiquement décrémenté et l'état précédent de la machine est restauré.

1.3.5.4 Changement rapide du contexte « Fast context switching »

C'est une technique qui permet de réduire le nombre de registres à sauvegarder dans une pile lorsqu'une exception survient. Un seul type d'exception peut être traité et c'est au logiciel de déterminer la cause de l'exception.

Lorsque le processeur passe d'un contexte à l'autre, et que le FSC est activé, un nouveau jeu de GPRs est activé et le compteur programme ainsi que l'adresse effective sont sauvegardés.

En utilisant le software (en provoquant un FCS), le traitement des interruptions et des threads (tâches) être gérés plus rapidement (car on évite la sauvegarde des registres internes dans la *pile*). Le hardware est capable d'exécuter le FCS en un cycle d'horloge.

Le contexte peut être changé durant une exception ou en utilisant le registre superviseur CXR (context register). Le CXR est le même pour tous les contextes.

Les fonctions du contexte principal sont:

- Passage d'un thread à l'autre.
- Manipulation des exceptions.
- Préparation, chargement, sauvegarde d'identificateurs de contexte de/à partir de la table CID.

1.3.5.5 Changement du contexte en mode superviseur

Le registre CXR est sur 32 bits; les 16 bits inférieurs représentent le registre contexte courant et les 16 bits supérieurs représentent le CID courant.

BIT	31-16	15-0
Identifier	CCID	CCRS
Reset	0	0

Figure 1.5: le registre CXR.

L'écriture dans le CCID induit le changement immédiat du contexte. La lecture du CCID donne le contexte courant.

Le CCRS a deux fonctions:

- Lorsqu'une exception survient, il contient (conserve) l'ancien CID.
- Il est utilisé pour accéder à d'autres registres contenant d'autres contextes.

1.3.5.6 Changement de contexte provoqué par une exception

Lorsque le FCS est activé, le registre CCID est copié dans le CCRS puis mis à 0, provoquant ainsi le passage au contexte principal.

Le CXR doit être sauvegardé dans un registre à usage général pour permettre si besoin est d'avoir plus d'exceptions imbriquées.

1.3.5.7 Accès aux registres d'autres contextes

L'accès se fait en mode superviseur est les instructions l.mtspr et l.mfspr sont utilisées pour accéder aux registres dits (shadow: ombres).

1.3.6 Organisation de la mémoire

L'architecture OpenRISC1000 spécifie un accès mémoire en désordre. En effet, c'est au programmeur de faire attention à l'ordre suivant lequel il effectue les lectures/écritures dans la mémoire. Cela permet un gain de performance surtout en environnement multiprocesseurs, mais ajoute une complexité supplémentaire pour le programmeur.

L'instruction l.msync permet la synchronisation de la mémoire. Le processeur s'assurera d'écrire toutes les données en mémoire avant de passer aux instructions suivantes.

1.3.7 Gestion de la mémoire

La gestion avancée de la mémoire est effectuée par une unité dite MMU (memory management unit).

La MMU spécifiée par l'architecture OpenRISC1000 est caractérisée par :

- Le support des adresses effectives sur 32 et 64 bits.
- Le support des adresses physiques allant jusqu'à 35 bits (32 Go).
- Trois tailles de pages mémoire:
  - ✔ Les pages de niveau 0. Traduites en utilisant le registre ATB (Area Translation Buffer).
  - ✔ Les pages de niveau 1. Traduites en utilisant le registre ATB.
  - ✔ Les pages de niveau 2. Traduites en utilisant le registre TLB (Translation Lookaside Buffer).
- Traduction des adresses en utilisant un, deux ou trois niveaux de tables de pages.
- Haute performance assurée par la protection des pages et le support de la mémoire virtuelle.
- Support du traitement multitâche SMT (Simultaneous Multi-Threading).

1.3.7.1 Utilité de la MMU

La fonction première de la MMU est de traduire les adresses effectives en adresses physiques. De plus, la MMU assure la protection des pages.

La MMU permet aussi d'adresser plus de mémoire physique que ne le permet la plage d'adresse effective. Cela se fait en prenant en compte en plus de l'adresse effective, les bits de

contexte (CID: Context ID) dans la calcul d'adresses.

La figure suivante résume la manière dont est traduite une adresse effective en adresse physique :

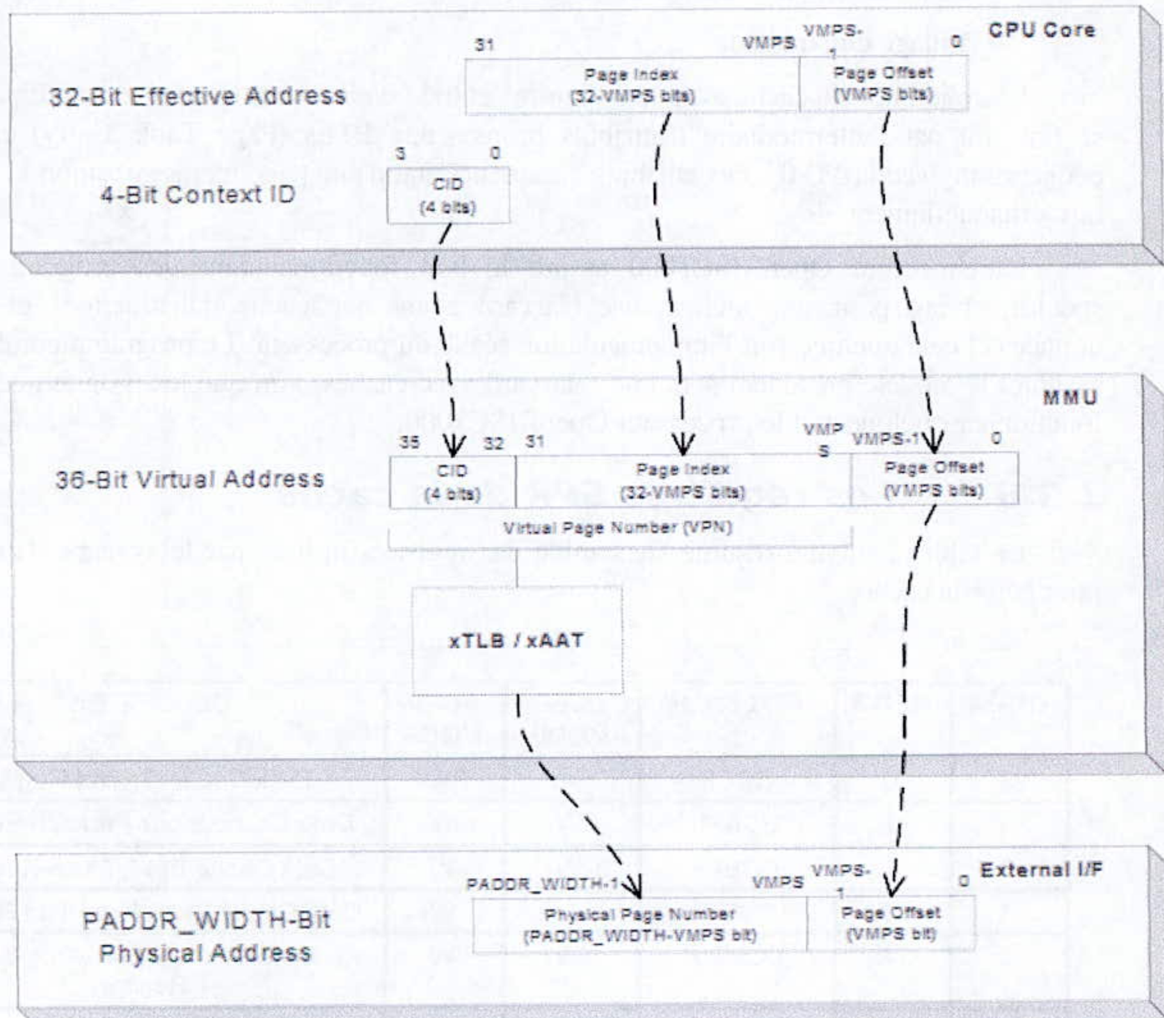


Figure 1.6: Transformation d'une adresse effective en adresse physique

C'est au système d'exploitation de mettre à jour les tables de correspondance entre adresses virtuelles (celles formées par les adresses effectives et les contextes) en adresses physiques.

1.3.8 La mémoire cache

La mémoire cache permet de précharger un bloc de code ou de données dans une mémoire interne au processeur avant même que le processeur n'y accède. Cela permet un gain important de performances venant du fait que cette mémoire est très rapide

La cohérence de la cache avec la mémoire doit être assurés par logiciel, cela se fait en

général en :

- Écriture du bloc en question.
- L'instruction l.sync permet d'attendre que la mise à jour de la mémoire ne finisse.
- Invalidation du bloc en cache (effacement de bloc).
- Vidage du pipeline.

La cohérence du cache avec la mémoire centrale dans un environnement multiprocesseurs se fait soit par l'intermédiaire d'attributs propres aux PTEs (Page Table Entry) et donc en coopération avec la MMU, ces attributs indiquent l'état d'une page mémoire, sinon la gestion se fait « manuellement ».

L'architecture OpenRISC1000 assure le bon fonctionnement des programmes écrits spécifiquement pour une architecture Harvard ayant une cache d'instructions et cache de données et cela quelque soit l'implémentation réelle du processeur. Le programmeur devra donc assumer le modèle précédemment cité (Harvard avec caches) afin que les programmes puissent fonctionner quelque soit le processeur OpenRISC1000.

1.3.8.1 Les registres SPR de la cache

Le tableau suivant résume l'ensemble des registres utilisés par le système d'exploitation pour gérer la cache.

GRP #	REG #	REG NAME	USER MODE	SUPV MODE	DESCRIPTION
3	0	DCCR	-	R/W	Data Cache Control Register
3	1	DCBPR	W	W	Data Cache Block Prefetch Register
3	2	DCBFR	W	W	Data Cache Block Flush Register
3	3	DCBIR	-	W	Data Cache Block Invalidate Register
3	4	DCBWR	W	W	Data Cache Block Write-back Register
3	5	DCBLR	-	W	Data Cache Block Lock Register
4	0	ICCR	-	R/W	Instruction Cache Control Register
4	1	ICBPR	W	W	Instruction Cache Block PreFetch Register
4	2	ICBIR	W	W	Instruction Cache Block Invalidate Register
4	3	ICBLR	-	W	Instruction Cache Block Lock Register

Table 1.7: Registres du cache.

Pour les implémentations utilisant une seule cache pour les données et les instructions, les registres de contrôle de données et des instructions sont fusionnés et valables en même temps en

tant que registres de données et d'instructions.

Pour gérer la cache, l'architecture possède deux registres qui ont pour rôle de contrôler la cache de données et d'instructions dénommés: « Data Cache Control Register » et « Instruction Cache Control Register ». Ce sont des registres de 32 bits et à usage spécial accessibles en mode superviseur par les instructions l.mtspr et l.mfspr.

1.3.9 Unité de débogage

L'unité de débogage permet d'aider les programmeurs à tester leurs programmes. L'unité de débogage contient des registres qui permettent de contrôler des points d'arrêt, des espions (watch points) et de l'exécution. Les fonctionnalités de l'unité de débogage sont :

- Elle est optionnelle.
- Huit jeux de registres de débogage/comparaison.
- Comparaison de conditions signées ou non pour l'adresse effective de l'instruction en cours de fetch, pour l'adresse effective de la donnée chargée/enregistrée ou pour la donnée chargée/enregistrée.
- Possibilité de combiner des conditions pour élaborer des espions complexes.
- Les espions peuvent générer des points d'arrêts.
- Comptage des espions afin de générer des espions supplémentaires.

1.3.10 Les compteurs de performance

Les compteurs de performance sont des compteurs qui permettent au développeur de compter certains évènements tels que le nombre de fetchs, le nombre de branchements et autres.

Ces compteurs sont très importants car ils permettent non seulement d'évaluer les performances et ainsi d'optimiser les routines systèmes au niveau algorithmique mais aussi d'évaluer les performances du processeur et donc de l'améliorer dans ses futures implémentations.

Les fonctionnalités implémentées sont :

- Huit compteurs définis par l'architecture.
- Huit compteurs personnalisables.
- Conditions de comptage programmables.

1.3.11 Unité de gestion d'énergie

La gestion efficace de l'énergie est devenue un enjeu majeur. Et comme OpenRISC1000 est destiné surtout aux environnements embarqués, le besoin d'optimiser la consommation d'énergie s'est fait sentir.

Les fonctions de l'unité de gestion de l'énergie sont:

- Fonctionnalité de ralentissement du processeur.
- Mode « sommeil léger » (doze mode).
- Mode veille.
- Suspension.
- Débranchement d'horloge dynamique.

La gestion d'énergie se fait par l'intermédiaire du registre superviseur PMR.

1.3.12 Contrôleur d'interruptions programmable

Le contrôleur d'interruptions programmable permet de gérer jusqu'à 32 sources d'interruptions externes. Cette unité est contrôlée par un registre qui contient le *masque* des interruptions activées (registre PICMR) ainsi qu'un registre d'état (PICSR) qui indique les interruptions actuellement activées.

1.3.13 Tick Timer

Cette unité contient un timer programmable, ces fonctions sont :

- Comptage jusqu'à 2^32 cycles d'horloge.
- Durée de comptage maximale de 2^28 cycles d'horloge entre les interruptions.
- Interruption masquable.
- Exécution unique, compteur redémarrable ou compteur continu.

Le timer est contrôlé par le registre en mode superviseur TTMR, la valeur courante du timer est sauvegardée dans le registre TTCR.

1.4 Implémentations existantes

Deux implémentations de la famille OpenRISC1000 sont connues. OpenRISC1200 développée par OpenCores et OpenRISC1500 développée par une équipe de chercheurs espagnols.

Pour notre part, nous avons utilisés OpenRISC1200 car OpenRISC1500 n'a été rendu public qu'assez récemment, de plus, les auteurs de ce dernier n'ont par encore mis à disposition le code source du core et des outils de développement.

1.4.1 OpenRISC1200

OpenRISC1200 [B10]est un processeur de la famille OpenRISC1000.

C'est un processeur scalaire 32 bits risc à architecture Harvard. Il comporte un cache de données et un cache d'instructions, la gestion avancée de la mémoire (MMU), des fonctions DSP de base (MAC), une unité de débogage, un timer, un contrôleur d'interruptions et une unité de gestion de l'énergie.

L'architecture globale d'OpenRisc1200 est donnée par la figure suivante:

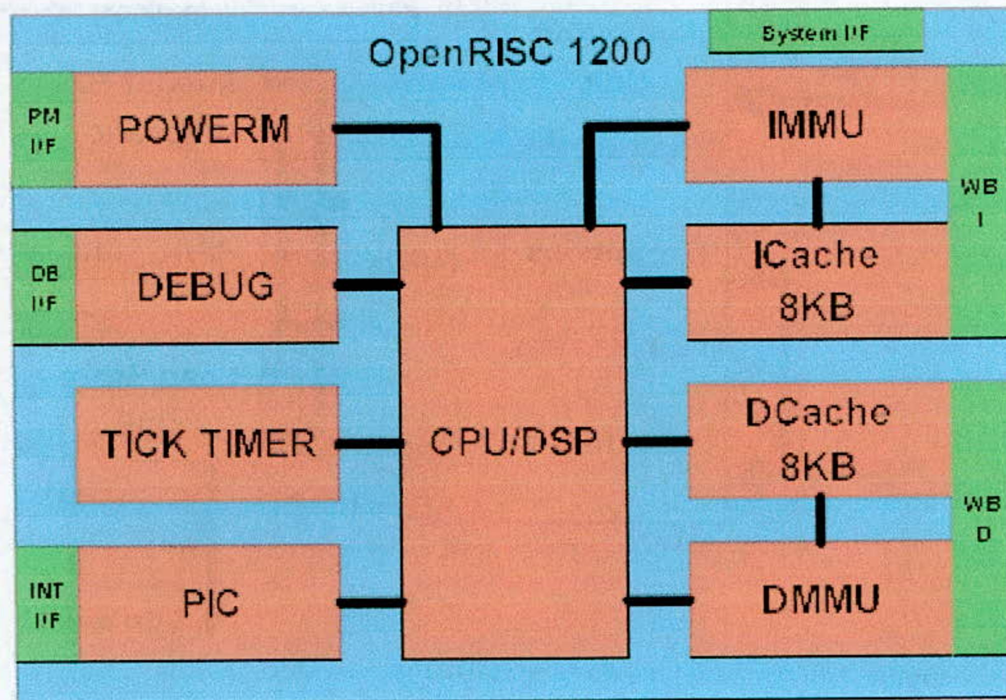


Figure 1.7: Architecture du processeur OR1200

La connexion du processeur avec le reste des périphériques se fait par l'intermédiaire du bus *WISHBONE*.

Le bloc CPU/DSP constitue le coeur du processeur, il est détaillé par la figure suivante:

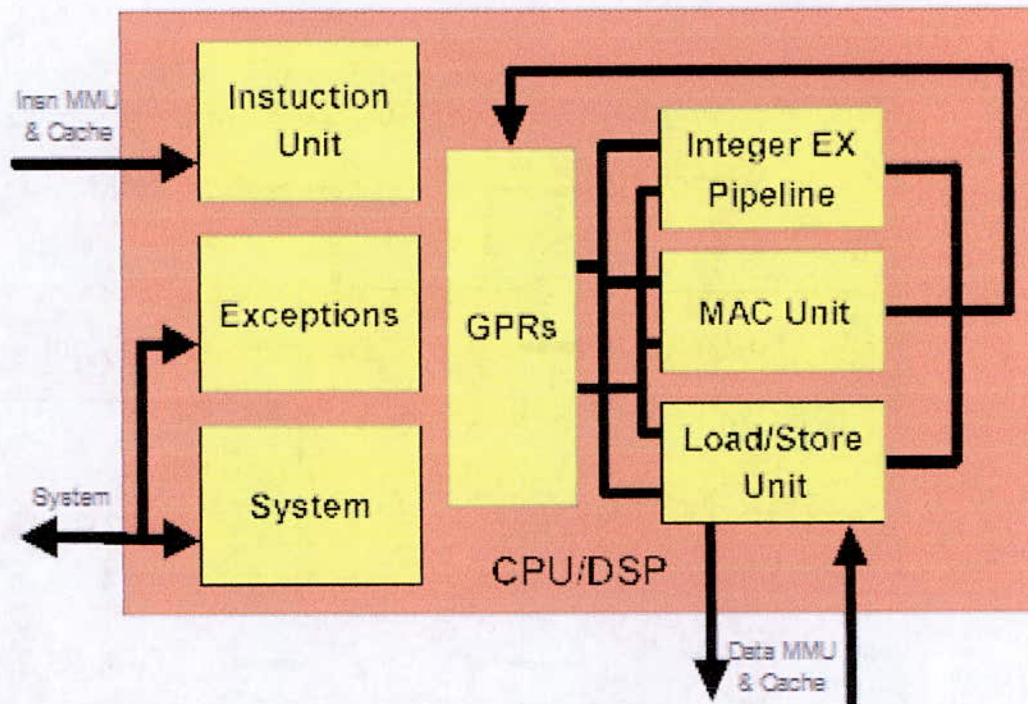


Figure 1.8: Architecture du bloc CPU/DSP

OR1200 contient 32 registres à usage général (GPR).

Le bloc Integer execution pipeline se charge de l'exécution :

- Des instructions arithmétiques.
- Des instructions de comparaison.
- Des instructions logiques.
- Des instructions de rotation et de décalage.

Le bloc Load/Store Unit de charge quand à lui de gérer les échanges avec les mémoires externes.

Le bloc MAC effectue l'opération de multiplication-accumulation. La multiplication se fait sur 32bits (entrées 32 bits et sortie 32 bits, les résultats sont tronqués sur 32 bits) et l'accumulation sur 48 bits. L'opération MAC se fait grâce à l'instruction l.mac.

Le bloc des exceptions gère les exceptions comme cela a été décrit dans l'architecture OpenRISC1000. Toutefois, les vecteurs d'exception sont différents, ils sont donnés par le tableau suivant:

Le bloc système contient les signaux système et les registres à usage spécial (SPRs) tels que le registre superviseur SR.

EXCEPTION TYPE	VECTOR OFFSET	CAUSING CONDITIONS
Reset	0x100	Caused by reset.
Bus Error	0x200	Caused by an attempt to access invalid physical address.
Data Page Fault	0x300	Generated artificially by DTLB miss exception handler when no matching PTE found in page tables or page protection violation for load/store operations.
Instruction Page Fault	0x400	Generated artificially by ITLB miss exception handler when no matching PTE found in page tables or page protection violation for instruction fetch.
Low Priority External Interrupt	0x500	Low priority external interrupt asserted.
Alignment	0x600	Load/store access to naturally not aligned location.
Illegal Instruction	0x700	Illegal instruction in the instruction stream.
High Priority External Interrupt	0x800	High priority external interrupt asserted.
D-TLB Miss	0x900	No matching entry in DTLB (DTLB miss).
I-TLB Miss	0xA00	No matching entry in ITLB (ITLB miss).
System Call	0xC00	System call initiated by software.
Breakpoint	0xD00	Initiated by the debug unit.

Table 1.8: vecteur des exceptions

Les instructions ORBIS32 sont implémentées dans le bloc Instructions.

Il est à noter que les instructions arithmétiques ne gèrent pas les dépassements (overflow), cela veut dire que le bit SR[V] n'a aucune signification et qu'il n'existe pas d'exception gérant les dépassements.

1.5 Évaluation du core

Une évaluation approfondie du core OpenRISC1000 a été faite [A04], cette évaluation à été faite sur la version OpenRISC1200 en utilisant l'outil OpenMore (MentorGraphics).

La vérification se fait en comparant la clarté de la documentation, le style de codage (ie. La manière dont est écrit le code source du core), les scripts de synthèse du core, la méthodologie de vérification.

Le core a obtenu une note globale de 73%, cela veut dire qu'il peut être réutilisé et qu'il est assez mature pour être utilisé dans des applications industrielles.

Il est clair que la qualité d'un core ne peut être améliorée qu'en ayant plusieurs implémentations de ce dernier. C'est pourquoi l'équipe qui s'est chargée de la vérification du

core a aussi décidé d'implémenter une nouvelle version (OpenRisc1500) de ce dernier en prenant en compte les résultats des tests effectués sur OpenRISC1200. Cela eut pour résultat un core ayant une vitesse 17% plus rapide que celle d'OpenRisc1200.

Chapitre 2: Les outils de développement

2.1 Introduction

Maintenant que nous avons vu les différentes fonctionnalités de la famille de processeurs OpenRISC1000, nous allons voir comment mettre en oeuvre des applications à base de ce dernier.

Lors de la réalisation d'une application ou d'un projet, l'utilisateur doit être capable de maîtriser les outils avec lesquels il doit travailler pour mener à bien son projet ou application.

Les concepteurs d'OpenRISC1000 ont choisis de porter les outils de développement GNU [B05] car ces derniers ont prouvés leur performances depuis plus de 15 ans (et même 20 ans pour certains outils).

Le portage des outils [W06], signifie l'ajout du support de l'architecture OpenRISC1000 à la suite logicielle GNU. L'utilisateur qui a déjà travaillé avec les outils GNU pourra aisément les utiliser pour développer des applications à base de processeurs OpenRISC1000.

Parmi ces outils on a: un compilateur C, un assembleur, un debugger, des utilitaires binaires « binutils », des librairies et d'autres outils développés pour améliorer la productivité, la portabilité, la flexibilité et les performances de l'application en question.

Il est important de noter que les systèmes d'exploitation uClinux, Linux, RTEMS et eCos ont tous été portés vers OpenRISC.

Il est à noter également la disponibilité d'un simulateur architectural pour systèmes à base d'OpenRISC1000.

Dans la prochaine section, nous allons voir les outils GNU utilisés pour développer des applications à base du processeur OpenRISC1000.

2.2 Les outils de développement GNU en résumé

La liste suivante résume l'ensemble des outils nécessaires au développement de projets:

- ***gcc***: le compilateur.
- ***cpp***: le préprocesseur.
- ***gdb***: le debugger.
- ***ld***: le linker.
- ***make***: programme contrôlant la compilation (compilation et recompilation

automatique).

- *as*: assembleur.

2.2.1 Les utilitaires

- *info*: documentation en ligne des outils GNU.
- *man*: ce sont des pages de documentation en ligne sous le standard Unix.
- *Addr2line*: conversion d'adresses en fichier et nombre de lignes.
- *ar*: crée, modifie et extraie à partir des archives de code objet (technique utilisée pour regrouper plusieurs fichiers objets).
- *nm*: affiche les symboles sous forme de liste à partir des fichiers objets.
- *objcopy*: copie et traduit les fichiers objets.
- *objdump*: affiche les informations à partir des fichiers objets.
- *randlib*: génère l'index au contenu de l'archive.
- *readelf*: affiche les informations concernant le format ELF des fichiers objets.
- *size*: affiche les fichiers avec leur taille et la taille totale sous forme de liste.
- *strings*: affiche sous forme de liste les caractères (strings) imprimables à partir des fichiers.
- *strip*: traitement des symboles.

2.3 Les librairies

Les librairies utilisées dans notre application:

- uClibc: qui est une version allégé de la librairie glibc.
- Linux regorge de librairies dans tous les domaines allant des librairies C de base en passant par des librairies système (accès aux fichiers, périphériques et autres), le traitement d'images (gdk), le traitement 3D (OpenGL), des librairies pour la création d'applications graphique (GTK, GNOME, wxWindows et bien d'autres).

2.4 gcc, the GNU Compiler Collection

gcc est une suite complète d'outils pour la compilation des programmes écrits en C, C++, objective C, ou d'autres langage installés (en se servant de *frontends*).

Le compilateur GNU utilise les utilitaires suivants:

- *cpp* : Le préprocesseur GNU. Il effectue les pré traitements sur tous les fichiers en-tête et exécute des macros avant la compilation.

- ***as*** : L' assembleur GNU. Il produit du code binaire à partir du code assembleur et met le résultat dans un fichier objet.
- ***ld*** : L'éditeur de liens GNU. Il traduit les symboles en adresses et relie le fichier de démarrage et les librairies au fichier objet.

L'appel au compilateur se fait par :

```
[shell]# gcc option1, option2....
```

Chaque option est une entrée dictant la façon dont le programme doit être compilé.

Plusieurs options permettent de générer des types de fichiers de sortie spécifiques compilés dont certains, d'autres pour le préprocesseur, pour le contrôle de l'assemblage, de l'édition des liens, du debuggage, de l'optimisation et d'autres fonctions spécifiques à la cible.

gcc reconnaît les extensions de fichier suivantes:

- .c: pour les codes sources écrits en C et qui doivent être pré traités.
- .C: pour les codes sources écrits en C++ et qui doivent être pré traités.
- .s: pour le code assembleur.
- .S :pour le code assembleur qui doit être pré traités.

Remarque:

Pour compiler du code source C++, on utilise ***g++***, cette compilation se fait d'une manière directe car il donne du code objet directement à partir du C++ sans passer par la traduction du code C++ en C puis de former le code objet à partir du code C. Cette manière de faire augmente les performances et simplifie le débogage.

La compilation peut avoir besoin de quatre étapes dans l'ordre suivant:

- Pré traitement (*cpp*).
- Compilation (***gcc***).
- Assemblage (***as***).
- Édition des liens (***ld***).

Les trois premières traitent un seul fichier à la fois, la compilation produit un ***fichier objet*** (cf 2.6.4 p42), l'assemblage établit la syntaxe que le compilateur attend pour les symboles et les constantes, les expressions et les directives. La dernière étape, édition des liens, complète la procédure de compilation, combinant l'ensemble des fichiers objet pour donner à la fin un fichier exécutable.

La figure suivante résume l'ensemble des opérations à effectuer pour obtenir un fichier exécutable à partir d'un fichier contenant du code source.

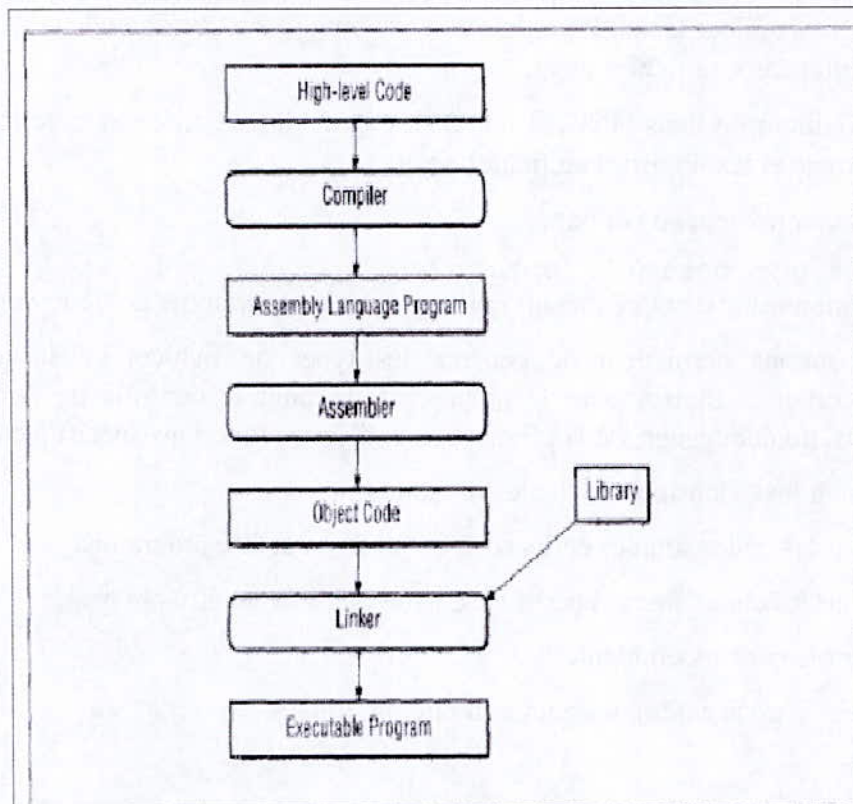


Figure 2.1: Transformations du code source.

2.5 cpp le préprocesseur GNU

cpp est un préprocesseur compatible C travaillant avec le compilateur ***gcc***, son rôle est de pré traiter les directives.

Pré traiter les directives consiste à chercher dans le programme les lignes commençant par le signe # et les traiter chacun selon le contenu, par exemple, on trouve des directives tel que: `#include, #ifdef et #define.`

L'appel au préprocesseur nécessite l'appel à ***gcc*** suivit de l'option `-E`. Le résultat du préprocesseur sera imprimé sur la sortie standard.

```
[shell]# gcc -E fichier_à_pré_traiter
```

Le préprocesseur permet d'effectuer les opérations suivantes:

- L'inclusion des fichiers headers (en-têtes) contenant toutes les déclaration dont le programme aura besoin.
- Développement des macros: ce sont des fragments de code C écrits sous forme de macro, leur utilisation consiste à remplacer une partie du code qui se répète par une chaîne de caractères, cette technique permet d'alléger le programme. Lors de pré traitement l'opération inverse est appliquée.
- Compilation conditionnelle: En utilisant des directives spéciales, des parties de

programme sont compilées ou pas selon la condition.

- Contrôle de ligne: On utilise un programme pour combiner ou réarranger les fichiers sources en un fichier intermédiaire qui sera par la suite compilé, pour fournir les lignes originales, on utilise le contrôle de ligne.

Il existe deux options pratiques pour l'assemblage des fichiers qui nécessitent un pré traitement de style C. Les deux options dépendent de l'utilisation du compilateur ***gcc*** au lieu d'appeler directement l'assembleur. Pour ce faire on peut utiliser une des deux méthodes:

- Renommer le fichier source en utilisant l'extension .S puis l'assembler.
- Spécifier le langage source explicitement en utilisant l'option : `-xassemembler-with-cpp`.

2.6 L'assembleur GNU as

L'assembleur ***as*** [B08] traduit du code écrit en langage assembleur contenu dans un fichier source en un fichier objet (binaire). Normalement on ne doit pas s'inquiéter pour son utilisation vu que le compilateur fait appel à l'assembleur de manière automatique. Cependant, si on doit créer un fichier source en langage assembleur, on doit faire appel directement à l'assembleur ***as***.

En réalité GNU ***as*** est une famille d'assembleurs. L'utilisation de l'assembleur sur une architecture donnée est pratiquement la même pour une autre architecture. Chaque version contient son format de fichier objet, ses *directives* (cf 2.6.10 p46)assembleur et sa syntaxe.

as est destiné à assembler le fichier de sortie produit par le compilateur C (gcc) puis utiliser le fichier assemblé par l'éditeur des liens ***ld*** pour produire un fichier exécutable.

La syntaxe utilisée par un assembleur peut être différente de celle issue d'un autre assembleur même si on utilise la même architecture.

2.6.1 Format des fichiers objet

L'assembleur GNU peut être configuré pour produire plusieurs types de formats de fichiers objet. Pour la plus part, ceci n'affecte pas la manière dont on écrit des programmes en langage assembleur, mais les directives pour le débogage ainsi que les symboles sont différents dans différent formats de fichiers.

2.6.2 La ligne de commande

La ligne de commande peut contenir des options et des noms de fichier. Les options peuvent apparaître dans l'ordre, avant ou après et entre les noms de fichiers. L'ordre des noms de fichiers est significatif.

N'importe quelle ligne de commande débutant par un signe (-) est une option. Chaque option change le comportement de l'assembleur ***as***. Une option ne peut changer l'effet d'une autre option. Certaines options ont besoin d'un seul fichier à la fois. Chaque fichier peut être suivi par une option ou par une autre commande.

2.6.3 Les fichiers d'entrée

Le programme source décrit l'entrée à exécuter par ***as***. Le programme peut être écrit sur un ou plusieurs fichiers; la manière dont le source est partagé entre différents fichiers ne change pas la signification du source.

En général, le source est une concaténation de texte dans tous les fichiers et dans l'ordre spécifié.

Chaque fois qu'on fait appel à ***as***, il ne peut assembler qu'un seul programme source, qui lui même est formé d'un ou plusieurs fichiers. Dans le cas où le programme source est vide, as produit un fichier objet vide.

Il y a deux manières pour localiser une ligne dans le/les fichiers d'entrée et peut être utilisé dans les messages d'erreur. La première façon est de faire référence au numéro de la ligne dans le fichier physique, l'autre façon est de faire référence au numéro de la ligne dans le fichier logique. Les fichiers physiques sont ceux nommés dans la ligne de commande donnés au as et les fichiers logiques sont simplement des noms déclarés explicitement par les directives assembleur, ils n'ont aucune relation avec les fichiers physiques. Les noms des fichiers logiques aident les messages d'erreur à refléter le fichier source original, lorsque celui ci est synthétisé à partir de plusieurs fichiers.

As est capable de comprendre les directives « # » destinées au préprocesseur ***cpp***.

2.6.4 les fichiers de sortie (objet)

Chaque fois que l'assembleur est mis en marche, il crée un fichier de sortie dit: objet. Ce sont des programmes assembleur traduits en code objet.

Le nom de fichier en sortie peut être par exemple fixé par défaut; il est soit *a.out* ou *b.out*. Il est possible de les nommer autrement en utilisant l'option -o.

Les fichiers objet sont des fichiers binaires résultants de la traduction du code assembleur en représentation binaire compréhensible par la machine cible et des informations permettant à l'éditeur des liens de générer un fichier exécutable. Aussi, ils contiennent des symboles liées au débogueur.

Il existe des programmes spéciaux pour manipuler les fichiers objet. Par exemple, ***objdump*** peut désassembler un fichier objet en un fichier contenant du code source (assembleur), aussi le programme ***ar*** permet de grouper plusieurs fichiers objet en un un seul fichier archive (librairie de fichiers objet).

2.6.5 Les erreurs et les avertissements

as peut écrire des erreurs et avertissements dans le fichier d'erreur standard (habituellement dans le terminal) ceci ne doit pas se produire lorsque le compilateur fait appel à ***as*** automatiquement. Les avertissements informent le programmeur des actions que l'assembleur effectue afin de garder le programme homogène en cas d'anomalies, contrairement aux erreurs

qui mentionnent la présence d'un problème grave qui stoppent l'opération d'assemblage.

Les avertissements (warning) ont la forme suivante:

```
nom_fichier:NNN:Warning corps du Message
```

NNN étant le numéro de la ligne. Si un fichier logique est donné, il sera utilisé pour le nom de fichier de sortie, sinon le nom du fichier d'entrée est utilisé. Si le numéro de ligne logique est donné, il sera utilisé pour calculer le nombre de ligne imprimé, autrement dit, la ligne actuelle du fichier source courant est imprimée. Le corps du message doit être explicatif.

Les messages d'erreur ont le format suivant:

```
nom_fichier:NNN:FATAL: corps du message d'erreur.
```

Le nom de fichier et le numéro de ligne sont tirés de la même façon que pour les avertissements. Le message actuel peut être plutôt moins que explicatif parce que plusieurs erreurs ne sont pas supposés avoir lieu.

2.6.6 Les options de la ligne de commande

Dans le cas où on fait appel à ***as*** à travers le compilateur C, on peut utiliser l'option `-Wa` pour passer des arguments à l'assembleur. Les arguments spécifiques à l'assembleur doivent être séparés par des virgules, par exemple:

```
[shell]# gcc -c -g -o -Wa,-alh,-L fichier.c
```

Dans ce cas on a deux options à passer à l'assembleur: `-alh` qui émet un listing sur la sortie standard contenant le code source ainsi que son équivalent e assembleur, l'option `-L` permet de retenir les symboles locaux dans la table des symboles.

Puisque plusieurs options sont passées automatiquement à l'assembleur, alors il n'est pas nécessaire d'utiliser à chaque fois le mécanisme `-Wa`. On peut utiliser l'option `-v` pour voir précisément quelle option à été passée à l'assembleur dans chaque étape de compilation.

Parmis d'autres options, on cite: `-a` (enables listing), `-f` (work Faster), `-I`(includeSearch Path), `-K` (Difference Tables), `-L` (Include Local Labels), `-L` (configuring listing output) et bien d'autres.

2.6.7 La syntaxe assembleur

Dans cette section, nous présenterons la syntaxe assembleur indépendante de la machine cible.

2.6.7.1 Pré traitement

Le préprocesseur interne de ***as***:

- Arrange et enlève les espace en plus. Il laisse une seule tabulation avant les mots-clé dans une ligne et transforme n'importe quel espace blanc en plus en un seul.
- Enlève tous les commentaires, les remplace par un seul espace ou bien par le nombre de lignes approprié.

- Transforme les constantes en valeurs numériques.

2.6.7.2 Espaces

L'espace blanc est utilisé pour séparer les symboles et rend le programme lisible pour l'être humain.

2.6.7.3 Les commentaires

Pour marquer un commentaire, on utilise les caractères suivants:

```
/* ceci est un commentaire */
```

2.6.7.4 Les expressions

Les expressions servent en général à évaluer la valeur d'un symbole.

Les expressions se terminent par un retour ligne.

Si on termine une expression par un EOF (End-Of-File), on provoque une erreur car le dernier caractère de n'importe quel fichier d'entrée doit être une nouvelle ligne.

Une expression vide est permise mais elle est ignorée.

2.6.7.5 Les constantes

Les constantes sont des nombres dont la valeur est connue par vérification. Les constantes peuvent être: un caractères, une chaîne de caractères, un nombre à virgule flottante, exprimés en octal, décimal ou en hexadécimal.

Il existe deux types de caractères:

- Les caractères s'écrivant sur un octet dont la valeur peut être utilisée dans des expressions numériques.
- Des constantes sous forme de chaînes de caractères appelées: « string » prenant plusieurs octets, leur valeur peut ne pas être utilisée dans les expressions.

Le caractère s'écrit entre quottes. Les caractères spéciaux doivent être précédés d'un antislash (ex: \n pour saut de ligne).

Si une nouvelle ligne suit la quotte (') alors elle est considérée comme un caractère.

L'assembleur ***as*** suppose que les caractères sont en code ASCII : par exemple: 'A est égal à 65, 'B est égal à 66 ....

Les strings sont écrits entre double quottes. Pour écrire des caractères spéciaux, on utilise un antislash suivi du caractère désiré. Le premier antislash est un espace permettant au as de l'interpréter entant que caractère littéraire. Parmi les caractères spéciaux on a : \b, \f, \r, \t , \x.

as distingue trois types de nombres constants selon la façon dont ils sont stockés dans la machine.

- Les entiers: sont similaires aux entiers décrits dans le langage c.

- Les bignums: sont des entiers stockés sur plus de 32 bits.
- Les flonums : sont des nombres à virgule flottante.

2.6.8 Les symboles

Les symboles sont très importants, il servent au programmeur pour nommer des adresses, *ld* utilise les symboles pour lier les objets entre eux et le debugger utilise les symboles pour debugger.

2.6.8.1 Les étiquettes

Les étiquettes (labels) sont des symboles suivit par (:). Le symbole représente la valeur courante du compteur de localisation (cf 2.7.4.5 p54). Dans le cas où l'on utilise le même symbole pour représenter deux localisations différentes, un message apparaîtra pour signaler que la première définition écrase toutes les autre définitions.

2.6.8.2 Attribution des valeurs aux symboles

Un symbole peut avoir une valeur arbitraire et ce en écrivant le symbole suivit de (=) suivit d'une expression. Ce-ci est équivalent à l'utilisation de la directive assembleur (.set)

2.6.8.3 Nom des symboles

Le nom du symbole commence par une lettre, par . ou _ certaines machines acceptent le signe $. Ces caractères sont suivis par une suite de chiffre, de lettre ou par le signe $. Il est à noter que certaines machines n'acceptent pas certaines notations, pour cela il faut consulter la documentation du constructeur.

Chaque symbole a un seul nom, chaque nom dans le programme assembleur renvoi un seul symbole.

2.6.8.4 Nom des symboles locaux

Les symboles locaux permettent au compilateur et au programmeur d'utiliser des noms temporaires, le symbole ainsi créé est unique.

Pour définir un symbole local on écrit un label de la forme `N:` où le N représente un nombre positif. Pour avoir la dernière définition du symbole, on doit écrire `Nb`, en utilisant le même nombre lors de la définition du label. Pour avoir la définition du prochain label local, on écrit `Nf` (f pour forward et b pour backward).

L'utilisation de ces étiquettes n'est pas limitée.

2.6.9 Les expressions

Les expressions spécifient une adresse ou une valeur numérique. Un espace blanc peut précéder une expression.

Le résultat d'une expression doit être un nombre absolu, ou un offset à l'intérieur d'une

section (cf 2.7.1 p47) particulière. Si l'expression n'est pas absolue et il n'y a pas d'informations suffisantes lorsque ***as*** vérifie sa section, un deuxième passage sur le programme source peut être nécessaire pour interpréter l'expression. Comme le deuxième passage sur le programme source n'est pas implémenté, ***as*** arrête l'exécution et affiche un message d'erreur.

2.6.9.1 Les expressions vides

Les expressions vides n'ont pas de valeur: c'est juste un espace blanc ou null. Partout où une expression absolue est demandée, on doit négliger l'expression le as suppose que la valeur est nulle. Ceci est compatible avec d'autres assembleurs.

2.6.9.2 Les expressions entières

Ce sont des expressions contenant un ou plusieurs arguments délimités par des opérateurs.

- Les arguments: ce sont des symboles, des nombres ou des sous-expressions. Dans un autre contexte, les arguments sont dits : opérandes arithmétiques. Pour éviter la confusion avec les opérandes de l'instruction du langage machine, on utilise le terme arguments pour faire référence à des parties de l'expression et le terme opérande pour faire référence aux opérandes de l'instruction machine. Les symboles sont évalués pour produire {section NNN}, où section peut être soit .text, .data ou .bss ou indéfinie. NNN est un nombre entier signé sur 32 bits complément à 2. Ce nombre peut être un bignum ou flonum, mais dans ce cas les 32 bits les plus faibles sont utilisés et ***as*** les considère comme un entier. Les sous expressions sont écrites entre deux parenthèses ou un opérateur (préfixe) suivi d'un argument.
- Les opérateurs: ce sont des fonctions arithmétiques, tel que + ou %. Les opérateurs (préfixe) sont suivis d'un argument. Les autres opérateurs sont entre leurs arguments. Les opérateurs peuvent être précédés ou suivis d'un espace.
  - ✔ Les opérateurs préfixe: as possède les opérateurs: - pour le complément à 2 (négation) et ~ pour l'inversion binaire.
  - ✔ Les opérateurs infix: ils prennent deux arguments, chacun sur un côté. Les opérateurs ont une priorité, mais les opérateurs ayant la même priorité sont exécutés de gauche à droite. Pour les opérateurs + et - , les deux arguments doivent être absolus et le résultat absolu.

2.6.10 Les directives assembleur

Les directives assembleur sont des commandes contenues à l'intérieur du code source, celles-ci peuvent contrôler la manière dont le fichier objet doit être généré. Elles sont aussi connues sous le nom de pseudo-opérations parce qu'elles peuvent ressembler à des commandes dans le langage assembleur de la machine cible.

Les directives assembleur commencent toujours par un point (.) et le reste c'est des lettres en minuscule. L'utilisation de ces dernières est très courante et vaste, par exemple : on l'utilise pour spécifier l'alignement, l'insertion de constantes dans le fichier de sortie (output) ...etc.

Par exemple, pour délimiter une section, on utilise la directive:

```
.section nom_de_section
```

2.6.10.1 Sections internes de l'assembleur

Ces sections sont valides uniquement pour l'utilisation interne du *as*. Au moment de l'exécution elles n'ont aucun effet. Elles sont utilisées pour permettre à chaque expression contenue dans le programme assembleur d'être une section ayant une adresse relative.

2.7 L'éditeur de liens GNU ld

L'édition des liens (effectuée par l'outil ***ld*** [B07]) est la dernière étape de la compilation, l'éditeur des liens (linker) combine plusieurs fichiers objet et archives et relocalise les données pour créer un programme exécutable.

Contrairement à d'autres éditeurs de liens et en plus de sa flexibilité, ***ld*** fournit des informations très utiles pour le diagnostic des erreurs. En effet, lorsque une erreur survient, certains éditeurs arrêtent immédiatement l'exécution, mais avec ***ld*** il est possible de continuer l'exécution et même d'obtenir un fichier de sortie malgré l'erreur.

L'édition de liens se fait:

- En résolvant les références entre les différents fichiers objet.
- En groupant les sections similaires contenues dans les fichiers objet en une seule place.
- En arrangeant les sections dans les adresses mémoires prévues.
- Et enfin; génération des informations nécessaires au démarrage. Ces information sont contenues dans l'en-tête du fichier.

2.7.1 Sections et relocalisation

Par définition une section est un ensemble d'adresses sans aucun espace (entre les adresses); toutes les données pointées par ces adresses sont traitées de la même manière, par exemple, elles peuvent être en mode lecture seule et dans ce cas on dira que la section est à lecture seule.

L'éditeur des liens ***ld***, lit plusieurs fichiers objet et combine leur contenu pour former un fichier exécutable. Lorsque ***as*** émet un fichier objet, le programme partiel est supposé commencer à l'adresse 0. ***ld*** attribue l'adresse finale pour le programme partiel et de cette façon, les différents programmes partiels ne vont pas se superposer.

2.7.1.1 Utilisation des sections

ld déplace les blocs d'octets du programme vers leur adresse de chargement (load-time adresse). Ces blocs glissent vers leur adresses en tant que unités rigides; ni leur longueur ni l'ordre des octets ne doit changer. De telles unités sont appelées: des sections.

On appelle relocalisation l'action d'assigner une adresse à une section.

Un fichier objet écrit par ***as*** a au moins trois sections, certaines peuvent être vide. Elles sont nommés:

- .text
- .data
- .bss

Dans le cas ou le format ELF ou COFF sont générés en sortie, ***as*** peut générer n'importe quelle section et ce, en utilisant la directive assembleur `.section`. Dans le cas où aucune directive (qui place les données dans une des sections) n'est utilisée alors les sections seront générées mais vides.

En général, à l'intérieur du fichier objet, la section `.text` aura comme adresse de début l'adresse 0, la section `.data` la suit. De la même façon, la section `.bss` suit la section data.

Pour que ***ld*** sache quelle donnée doit il changer lorsque les sections sont relocalisées et comment changer cette donnée, ***as*** écrit en plus dans le fichier objet des informations destinées à la relocalisation. Pour pouvoir relocaliser, ***ld*** doit donc pouvoir connaître à chaque fois qu'une adresse est mentionnée:

- Où dans le fichier objet commence la référence à l'adresse?
- La longueur en octets de cette référence?
- L'adresse pointe quelle section? Quelle est la valeur numérique de l'adresse de début et de fin de section?
- Est ce que l'adresse de référence est relative au compteur programme?

En réalité chaque adresse ***as*** est toujours exprimée par : (section) + (offset à l'intérieur de la section).

En plus de la connaissance des trois sections, il est nécessaire de connaître la section dite: absolue. Lorsque ***ld*** mélange les programmes partiels, les adresses contenues dans la section absolue ne changent pas. La section indefinie rassemble quand à elle le reste du code. En chaque adresse dont la section n'est pas connue au moment de l'assemblage est par défaut placée dans {indéfinie -U} où le -U signifie: qu'elle sera remplie après.

Certaines sections sont manipulées par ***ld***; d'autres sont inventées pour être utilisée avec ***as*** et elles n'ont aucun sens qu'au moment de l'assemblage.

2.7.1.2 Les sections du linker ld

ld travaille avec quatre types de sections qui sont:

- Les sections nommées: text et data organisent le programme. As et ld les traitent séparément mais les considèrent comme égales. Chaque chose appliquée à une section est valable pour l'autre. Habituellement lorsque le programme est en

exécution, la section texte est inaltérable. Parfois elle est partagée entre plusieurs processus car elle contient des instructions, des constantes et des références. La section de donnée du programme en cours d'exécution est altérable, par exemple les variables C seront stockés dans cette section.

- Section .bss: lorsque le programme commence à s'exécuter, elle contient des octets mis à 0. Elle est utilisée pour les variables non initialisées ou pour l'espace commun de stockage. Dans cette section la longueur de la section .bss de chaque programme partiel est importante. Puisque cette section contient des zéros, on n'a pas besoin de la stocker explicitement dans le fichier exécutable.
- Section absolue: l'adresse 0 de cette section est toujours relocalisée à l'adresse 0. Ceci est intéressent dans le cas où on veut que ***ld*** ne change pas l'adresse lors de la relocalisation.
- Section indefinie: C'est une section qui regroupe tout ce qui n'appartient pas aux autres sections.

2.7.1.3 Attributs des sections

Chaque section dans un fichier objet a son propre nom et sa propre taille. La plupart des sections ont des blocs de données associés connus sous le nom de : « contenu de la section ».

Une section peut être:

- Allouable : c'est à dire que l'espace mémoire doit être réservé exclusivement à cette section au début de l'exécution du fichier.
- Chargeable : c'est à dire que son contenu doit être chargé dans la mémoire lorsque l'exécution commence.

Une section qui est « allouable » et pas « chargeable » aura une zone mémoire remplie avec des zéro.

Une section qui est ni chargeable ni allouable contient typiquement des informations de debuggage.

Chaque sortie allouable ou chargeable a deux adresses associées. La première est l'adresse mémoire virtuelle VMA (Virtual Memory Adress) qui est l'adresse de la section lorsque l'exécutable commence à s'exécuter. La deuxième est l'adresse mémoire de chargement LMA (Load Memory Adress) qui est l'adresse dans la mémoire où la section doit être chargée.

Dans la plus part des cas, les deux adresses sont les mêmes. Elle peuvent être différentes dans le cas où la section des données est chargée dans la ROM mais qui doit être copiée dans dans la RAM lorsque le programme s'exécute. Dans ce cas, l'adresse de la ROM est une LMA et l'adresse de la RAM est une VMA .

Chaque fichier objet possède une liste de symboles, connue sous le nom de: table des symboles. Un symbole peut être défini ou non. Chaque symbole possède un nom et une adresse. Si on compile un programme en C/ C++ dans un fichier objet, on aura la définition des

symboles pour chaque fonction et variable globale ou statique.

Chaque fonction ou variable globale indéfinie, référencée par le fichier d'entrée va être un symbole indéfini. On peut obtenir la table des symboles d'un fichier objet en utilisant l'utilitaire binaire ***nm*** , ou en utilisant l'utilitaire binaire ***objdump*** avec l'option `-t`.

2.7.2 Les options de la ligne de commande

Pour appeler ld, on peut taper la commande suivante:

```
[shell]# ld -o output nom_de_fichier -lc
```

Dans cet exemple, on demande à ***ld*** de produire un fichier de sortie nommé output à partir du fichier nom_de_fichier et du fichier *libc.a* (librairie).

Certaines options peuvent être spécifiées à n'importe quel point de la ligne de commande. Cependant, les options traitants les fichiers tel que `-l` ou `-T`, provoquent la lecture du fichier au point où l'option a été utilisée (dans la ligne de commande) relativement aux fichiers objet et aux options des fichiers.

Les arguments sont des fichiers objet ou des archives qui vont être reliées entre eux. Ils peuvent suivre, précéder ou se mélanger avec les options de la ligne de commande, exception faite pour les arguments des fichiers objet; ils ne peuvent être placés entre les options et leurs arguments.

Si aucun des fichiers d'entrée n'est binaire, alors ld ne produit aucun fichier de sortie et affichera le message suivant:

```
4. No input files.
```

Si ***ld*** n'arrive pas à déterminer le format d'un des fichiers alors il le considérera comme étant un script, ce dernier enrichira le script par défaut. Si on veut utiliser un script personnalisé au lieu du script par défaut, il faut utiliser l'option `-T`.

Les options s'écrivant avec une seule lettre, leurs arguments doivent soit suivre l'option sans laisser d'espace, ou de les écrire séparément avec un espace blanc.

Pour les options dont le nom est une suite de lettres, il doivent être précédé par un tiret (-) ou deux (--). Pour éviter la confusion avec l'option (– o), les options à plusieurs lettres et commençant par la lettre 'o' doivent être précédées par deux tirets (--).

Les arguments des options à plusieurs lettres doivent être séparés du nom de l'option par un signe égal (=) ou séparé par un espace blanc.

Remarque: dans le cas où ***ld*** est appelé à travers le compilateur ***gcc***, alors toutes les options de l'éditeur des liens (dans la ligne de commande) doivent être précédées par `-Wl` par exemple :

```
[shell]# gcc -Wl,--startgroup foo.o bar.o -Wl,--end group
```

Il existe une multitude d'options spécifiques à ***ld***, pour plus d'information, voir la documentation.

2.7.3 Variables d'environnement

On peut changer le comportement de ld par les variables d'environnement suivantes: GNUTARGET, LDEMULATION et COLLECT_NO_DEMANGLE.

La première détermine le format du fichier objet d'entrée dans le cas où on n'utilise pas l'option `-b`.

S'il n y a pas GNUTARGET dans l'environnement, ld utilise le format par défaut de la cible.

LDEMULATION détermine l'émulation par défaut dans le cas où l'option `-m` n'est pas utilisée. L'émulation peut affecter le comportement de ***ld*** et particulièrement le script par défaut.

En temps normal, ***ld*** va réarranger les symboles par défaut. Cependant si COLLECT_NO_DEMANGLE est présente dans l'environnement alors les symboles ne seront pas réarrangés par défaut.

2.7.4 Les script de l'éditeur des liens

Chaque édition des liens est contrôlée par un script, ce dernier est écrit dans un langage spécifique.

L'objectif de ce script est de décrire la manière dont les fichiers d'entrée doivent être organisés dans le fichier de sortie ainsi que l'organisation mémoire. La plupart des scripts peuvent faire plus que ça.

Toujours, l'éditeur des liens utilise le script. Si on ajoute pas notre script, il va donc prendre le script par défaut. Pour afficher le contenu de ce script, il suffit d'ajouter l'option `--verbose` dans la ligne de commande. Les options `-r` et `-N` permettent d'augmenter ce script.

Lorsqu'on utilise un script personnalisé (par l'option -T), le script par défaut sera remplacé par celui-ci.

2.7.4.1 *Le format du script linker*

Les scripts sont des fichiers texte. Ils sont composés d'une série de commandes. Chaque commande est un mot clé probablement suivi par des arguments ou des affectations de valeurs pour les symboles. Les commandes sont séparées par des (;), l'espace est ignoré.

Les chaînes de caractères tel que les noms des fichiers et les formats peuvent être écrits directement, il est possible d'ajouter des commentaires dans le script de la même manière que dans les programmes C c'est à dire: /* et */, les commentaires sont équivalents à des espaces blancs.

2.7.4.2 *Exemple de script*

Le plus simple des script contient une seul commande: `SECTION` on utilise cette commande pour décrire l'organisation mémoire dans le fichier de sortie.

```
SECTIONS
```

```
{
        .=0x10000;
        .text:{*(.text)}
        .=0x8000000;
        .data:{*(.data)}
        .bss:{*(.bss)}
}
```

L'éditeur de liens se comportera comme suit:

- Assignation de la valeur 0x10000 au compteur de localisation.
- Copie de toutes les sections .text des fichiers d'entrée vers la section .text du fichier de sortie. La section .text du fichier de sortie débutera à l'adresse 0x10000 puisque le compteur de localisation à été initialisé à cette valeur.
- Assignation de la valeur 0x8000000 au compteur de localisation.
- Copie des sections .data des fichiers d'entrée vers la section .data du fichier de sortie. La section data du fichier de sortie débutera à l'adresse 0x8000000.
- Copie des section .bss des fichiers d'entrée vers le fichier de sortie. La section .bss du fichier de sortie se trouvera à la suite de la section .data.

2.7.4.3 Les commandes du script

- Insertion du point d'entrée : La première instruction à exécuter dans un programme est appelée: point d'entrée (entry point). On peut utiliser la commande `ENTRY` pour fixer le point d'entrée. L'argument est un nom de symbole: `ENTRY (symbole)`.
- Les commandes traitants des fichiers: il y a plusieurs commandes, parmi elles on cite: `INCLUDE, GROUP, OUPUT, SEARCH, STARTUP.`
- Les commandes traitants les formats de fichiers objet: ici on a deux commandes: `OUTPUT_FORMAT(bfdname)` et `OUTPUT_FORMAT(default, big, little)`.
- Attribution des valeurs aux symboles: cette opération définit le symbole entant que symbole global. Les opérateurs utilisés dans ce cas sont similaires à ceux utilisés dans le langage C, par exemple :

```
symbole += expression;
```

2.7.4.3.1 La commande sections

Le format de cette commande est:

```
SECTIONS
{
```

```
section-commande
section-commande
}
```

Chaque `sections-command` peut être :

- La commande `ENTRY`.
- Attribution des valeurs aux symboles.
- Description d'une section de sortie.
- Description overlay (que nous n'avons pas traitée).

La commande `ENTRY` et l'assignment des symboles sont permis à l'intérieur de la commande `SECTIONS` à fin de pouvoir utiliser le compteur de localisation comme argument. Ceci rend la compréhension du script ***ld*** facile.

Dans le cas où on n'utilise pas la commande `SECTION` dans le script, ***ld*** va placer chaque section d'entrée dans une section de sortie ayant le même nom que la première et dans le même ordre d'apparition (dans la section d'entrée). Par exemple, si toutes les sections sont présentes dans le fichier d'entrée, l'ordre des sections dans le fichier de sortie va correspondre à celui du fichier d'entrée et la première section va être placée à l'adresse zéro.

2.7.4.3.2 La commande MEMORY

La configuration par défaut de ***ld*** permet l'allocation de tout l'espace mémoire. Pour changer cette dernière, la commande `MEMORY` est utilisée.

Cette commande décrie l'emplacement et la taille des blocs mémoire de la cible. On peut l'utiliser pour dire quelle région de la mémoire peut être utilisée par ***ld*** et quelle région doit il éviter. On peut par la suite attribuer des sections à une région mémoire.

Pour fixer les adresses des sections, ***ld*** se base sur les régions mémoire et prévient l'utilisateur lorsqu'une des régions est pleine. ***ld*** ne va pas déplacer les sections pour les mettre dans une région mémoire libre.

Le script ne peut avoir qu'une seule commande MEMORY. Cependant, on peut définir autant de blocs mémoire qu'il le faut.

La syntaxe de la commande `MEMORY` est la suivante:

```
MEMORY
{
 nom [(attr)] : ORIGIN , LENGTH = len
 ...
}
```

- `nom`: est le nom utilisé dans le script faisant référence à une région mémoire. Le nom

de la région n'a aucune signification en dehors du script. Les nom des régions sont sauvegardés dans des espaces de noms différents. Cette procédure ne doit pas générer des conflits avec: les noms de symboles, les noms de fichiers ou les noms de sections. Chaque région mémoire doit avoir son propre nom.

- `attr`: est une liste des attributs optionnels spécifiant l'utilisation des régions mémoire particulières pour les sections d'entrée et qui ne sont pas organisées dans le script. Dans le cas où on ne spécifie pas une section de sortie pour une section d'entrée donnée, ***ld*** va créer une section de sortie ayant le même nom que celle de l'entrée. Si on définit les attribues d'une région, ***ld*** va les utiliser dans le but de sélectionner une région mémoire pour la section de sortie ainsi créée.
- `ORIGIN`: est une expression pour le début d'adresse dans la région mémoire. L'expression doit être évaluée à une constante avant que l'opération d'allocation mémoire ne soit exécutée, on ne doit donc pas utiliser des symboles définis dans des sections.
- `len`: est une expression définissant la taille en octet de la région mémoire. De la même façon que pour `ORIGIN`, `len` doit être évaluée avant l'exécution de l'opération d'allocation mémoire.

2.7.4.4 Les expressions dans le script

La syntaxe des expressions dans le langage du script (spécifique au ***ld***) est identique à celle du langage C. Toutes les expressions sont évaluées entant qu'entiers de 32 bits dans le cas où l'hôte et la cible sont sur 32 bits.

On peut utiliser les valeurs des symboles dans les expressions.

2.7.4.5 Le compteur de localisation

La variable spéciale à l'éditeur des liens `.` contient toujours la valeur courante du compteur de localisation. Puisque `.` fait référence à une adresse ou localisation dans les sections de sortie, elle peut apparaître uniquement dans une expression à l'intérieur d'une commande `SECTION`. Le symbole peut apparaître à n'importe quelle position où le symbole est autorisé dans une expression.

Attribuer une valeur au `.` va déplacer le compteur de localisation. Ceci peut être utilisé pour créer des vides dans les sections de sortie. Le compteur de localisation ne doit jamais être déplacé en arrière.

2.7.4.6 Autres commandes

ld définit plusieurs fonctions spéciales utilisées dans le script.

Parmi ces fonctions on a:

- `ALIGN (exp)`: elle retourne le compteur de localisation courant (.) aligné avec la prochaine expression. Cette fonction ne change pas la valeur du compteur, mais

exécute des opérations arithmétiques sur lui.

- `DEFINED (symbol)`: elle retourne 1 si le symbole est définit dans la table des symboles du script, sinon, elle renvoie 0. On peut utiliser cette fonction pour donner des valeurs par défaut aux symboles.
- `Sizeof (section)`: si une section est allouée dans la mémoire alors, sizeof retourne la taille en octet de cette dernière. Si celle ci n'est pas encore allouée en mémoire lors de l'évaluation de cette fonction, alors ld retourne une erreur.

2.8 Gcov, outil de test de couverture

gcov est un outil de test , il permet de voir le degré de couverture d'un programme; il permet d'analyser les blocs essentiels du programme et ce en enregistrant combien de fois un bloc est exécuté. Cette information permet de déterminer les sections du code qui ne s'exécutent jamais et de déterminer le chemin critique (critical path).

2.9 L'outil de debogage, gdb

gdb est le debugger GNU, son principal rôle est de permettre l'arrêt du programme avant sa fin. Si le programme se plante, le debugger permet de voir pourquoi le programme a échoué.

Pour debbuger un fichier nommé: *file_name.c*, il doit être compilé en utilisant la commande suivante:

```
[shell]# gcc -g file_name.c
```

où l'option `-g` produit les informations de debogage. Ensuite on met en marche le debugger.

Pour insérer des points d'arrêt on utilise (à partir de l'invite ***gdb***) la commande: `breakpoint`.

Pour naviguer à l'intérieur du programme, on utilise la commande: `step` ou `next`.

Le debugger permet de traiter, signaler et tracer des informations et d'autres donnée dans le programme.

Chaque fois que le programme fait appel à une fonction, un bloc de donnée représentant l'emplacement de l'appel, les arguments et les variables locales de la fonction est généré. Aussi, avec le debugger, on peut examiner la pile (stack) dans le but de faire fonctionner le programme en question.

2.10 L'outil de recompilation, Make

make est un outil permettant la recompilation des programmes, il détermine d'abord quelles partie d'un gros programme a besoin d'être recompilée puis exécute les commandes nécessaires à la recompilation.

make est conforme au standard IEEE 1003.2-1992 (POSIX.2) et il est compatible avec

n'importe quel langage de programmation dont le compilateur peut fonctionner en mode ligne de commande et peut accepter des arguments en entrée à partir du shell.

make n'est pas limité uniquement a développer des programmes; il peut être utilisé pour n'importe quelle tâche dans la quelle certains fichiers doivent être mis à jour automatiquement à chaque fois qu'un des fichiers change.

Pour utiliser ***make***, on doit écrire un fichier nommé: *Makefile* décrivant les relations entre les fichiers du programme et fournissant les commandes mettant à jour chaque fichier. Typiquement, dans un programme, le fichier exécutable est mis à jour à partir des fichiers objet, qui à leur tour sont crées en compilant les fichiers source.

Lorsqu'on utilise ***make*** pour recompiler l'exécutable, le résultat peut changer les fichiers sources. Par exemple, si un fichier en-tête change, il sera nécessaire de recompiler chaque fichier source incluant cet en-tête, puis de recompiler l'exécutable final.

make utilise la base de donnée décrite par le *Makefile* et les derniers fichiers modifiés pour décider quels sont les fichiers à mettre à jour.

Makefile exprime donc les règles expliquant comment et quand recompiler certains fichiers qui sont la cible d'une certaine règle.

Un *Makefile* simple a la forme suivante:

```
target ... : dependency .....command
```

Où *target* est le nom du fichier qu'un programme génère, par exemple un exécutable ou un fichier objet. *target* peut être aussi le nom d'une action à effectuer, tel est le cas pour la commande clean.

dependency: ce sont les fichiers utilisés en entrée pour créer la cible. La cible dépend parfois de plusieurs fichiers. La commande est activée par ***make*** lorsque les dépendances sont changées.

Une règle peut avoir plus qu'une commande, chaque commande devant être placée dans une seule ligne.

Certaines règles spécifient les commandes pour une cible donnée qui ne nécessite pas de dépendance. Une règle peut aussi expliquer comment et quand activer une commande.

Notons que chaque ligne de commande dans le *Makefile* doit avoir une tabulation au début de la ligne.

2.11 Binutils, les outils binaires

C'est un ensemble d'utilitaire de manipulation des fichiers binaires. Binutils [B06] inclut ***ar***, ***nm***, ***objcopy***, ***objdump***, ***randlib***, ***size***, ***strings*** et ***strips*** ». Pour les cibles utilisant le format de fichier ELF, il existe aussi un outil nommé: ***readelf***. Les utilitaires : ***addr2line***, ***winders*** et ***dlltool*** sont utilisés avec *cygwin* et permettent le portage des applications vers la plateforme win32.

Les utilitaires les plus importants sont:

- ***objcopy***: qui est un outil de conversion d'objet et de fichiers exécutables. Il peut ajouter et enlever des sections et des symboles, mais généralement la fonction la plus utilisée est le changement le format du fichier. Par exemple, on peut convertir des exécutables au format ELF ou COFF en un S-record ou I-HEX. Ces deux formats sont parfois utilisés pour construire une image ROM pour les exécutables indépendants et les systèmes embarqués.
- ***Objdump***: c'est un outil pour afficher les informations concernant un fichier exécutable ou objet. Il peut afficher la table des symboles, les sections, les en-têtes et il peut aussi agir comme un dés-assembleur. ***Objdump*** reconnaît les archives et les librairie, il peut être utilisé pour afficher des informations sur tous les fichiers objet contenus dans ces dernières.

2.12 Le simulateur architectural

Le simulateur architectural est un simulateur pour OpenRISC1200. Sa particularité est le fait qu'il simule non seulement le processeur mais aussi les périphériques l'entourant.

Le simulateur architectural est donc capable de simuler :

- Le processeur OpenRISC1200.
- Le contrôleur de mémoire.
- Les UART (interfaces série).
- DMA (Direct Memory Access).
- Ethernet (Interface réseau).
- GPIO (General Purpose Input/Output).
- VGA/LCD (écrans graphiques standards et à cristaux liquides).
- Clavier PS2.
- ATA (contrôleurs de disquette/disque dur).

De plus, le simulateur permet de spécifier le memory map, de simuler un serveur JTAG (standard IEEE qui permet la vérification in-situ) permettant ainsi de connecter le simulateur à un debugger comme s'il s'agissait d'une vraie carte. Il inclut aussi une API (Application Programming Interface) pour la vérification.

Il est important de noter que le simulateur n'est pas documenté car, comme la plupart du port GNU, il est toujours en cours de développement. C'est pourquoi, nous avons eu très souvent à lire le code source de ce dernier afin de comprendre son fonctionnement, ce qui constitue un facteur ralentissant dans le travail à effectuer.

Le simulateur est contrôlé par un fichier *.cfg* qui contient la définition des différents périphériques et des unités internes du processeur ainsi que le memory map.

2.13 Installation des outils de développement

2.13.1 Introduction

La première étape d'un travail d'implémentation d'un système à base du processeur OpenRISC est l'écriture des programmes, la compilation puis la validation de la partie soft.

Pour cela, il faut disposer d'une suite d'outils:

- Un éditeur de texte. Pour l'instant: ***emacs***.
- Un compilateur, Linker... : ***gcc***
- Des outils pour manipuler les fichiers binaires: Binutils.
- Un système d'exploitation pour le système cible: uClinux.
- Une librairie C: uClibc.
- Un simulateur. Or1ksim qui est un simulateur architectural.

Pour mettre en place une telle infrastructure logicielle, ils est indispensable de bien connaître la philosophie des différents outils, l'interaction des uns avec les autres. Ils est en conséquence primordial de respecter l'ordre d'installation suivant:

1. Binutils.
2. gcc.
3. gdb.
4. Or1ksim.
5. uClinux.
6. uClibc.
7. Recompilation de gcc pour qu'il prenne en compte uClibc.

Il est à noter qu'il est préférable d'utiliser gcc 2.95 pour que la compilation de tous les outils se fasse correctement.

2.13.2 Installation de gcc-2.95

Si *gcc-2.95* est installé sur votre système, vous pouvez passer à la suite.

Pour savoir quelle version de ***gcc*** est installée sur votre système: tapez la commande suivante:

```
[shell]# gcc --version
```

pour installer gcc 2.95, tapez les commandes suivante dans une invite shell:

```
[shell]# nice  ./configure  --prefix=/opt/gcc-295  -program-transform-name='s/\\(.*\\)/\\1-295/'
```

```
[shell]# nice make bootstrap
[shell]# nice make install
[shell]# export PATH=/opt/gcc-295/bin:$PATH
```

Pour utiliser le compilateur, il suffit de modifier la variable d'environnement CC qui contiendra la valeur ***gcc-295***.

2.13.3 Installation de binutils

Binutils est un ensemble d'outils permettant la manipulation des fichiers binaires (ex. fichiers objets, executables). Il inclus par exemple un assembleurs, un désassembleur, un éditeur de liens.

Pour installer binutils, tapez les commandes suivantes:

```
[shell]# nice   env   CC=gcc-295   ../binutils/configure   --
target=or32-uclinux -prefix=/opt/or32-uclinux
[shell]# nice /usr/bin/make all install
```

2.13.4 Installation de gcc pour or1k

Gcc est un compilateur C/C++ standard et très puissant. Il permet de faire du « cross compiling » c'est à dire de compiler sur une architecture autre que l'architecture cible.

Gcc peut compiler pour des dizaines d'architectures cibles et supporte (en plus du C) les langages les plus connus.

Nous allons installer gcc 3.2.3 avec comme plate forme cible or32-uclinux, cela veut dire que le compilateur donnera en sortie du code exécutable par le processeur OpenRISC version 32 bits et tournant sous uClinux.

Pour faire cela, il suffit de taper les commandes:

```
[shell]# nice env CC=gcc-295 \
../gcc-3.2.3/configure --target=or32-uclinux \
--with-gnu-as --with-gnu-ld --verbose \
--enable-threads --prefix=$INSTALL_PREFIX \
--local-prefix=/opt/or32-uclinux/or32-uclinux \
--enable-languages="c"
[shell]# nice /usr/bin/make all install
```

2.13.5 Installation de gdb 5.0

Gdb est un débugger open source. Il permet de tracer l'exécution d'un programme, d'espionner la mémoire, d'afficher la pile des appels et bien d'autres fonctions d'aide au déboguage.

Gdb peut exécuter des programmes sur la machine hôte, sur une machine distante grâce à des interfaces de débogage (ex. JTAG) ou alors s'exécutant sur un simulateur.

```
[shell]# env CC=gcc-295 ../gdb-5.0/configure \
--target=or32-uclinux -prefix=$INSTALL_PREFIX
[shell]# nice /usr/bin/make all install
```

2.13.6 installation de or1ksim

Or1ksim est un simulateur architectural pour systèmes à base de processeur OpenRISC. Cela veut dire qu'il est capable de simuler non seulement le processeur mais aussi les périphériques qui l'entourent. On peut donc valider une solution entière sans avoir à concevoir un prototype.

Pour installer or1ksim, exécutez les commandes suivantes:

```
[shell]# autoconf
[shell]# automake-1.6
[shell]# env CC=gcc-295 ../or1ksim/configure \
--target=or32-uclinux -prefix=/opt/or32-uclinux
[shell]# nice /usr/bin/make all install
```

2.13.7 installation d'uClinux

uClinux est un système d'exploitation pour processeurs dépourvus de MMU. Uclinux veut dire microcontroller linux.

Pour l'installer, tapez les commandes suivantes:

```
[shell]# cd uclinux/uClinux-2.0.x
[shell]# sed -e 's/^LIBGCC/#LIBGCC/; \
/^#LIBGCC/ a \
LIBGCC          =          '$INSTALL_PREFIX'\/lib\/gcc-lib\/or32-
uclinux\/3.2.3\/libgcc.a' \
< arch/or32/Rules.make \
> arch/or32/Rules.make.edited
[shell]# mv arch/or32/Rules.make.edited arch/or32/Rules.make
[shell]# sed  -e  's/^CONFIG_NET/#CONFIG_NET/;  /^#CONFIG_NET/
a \
CONFIG_NET=n'          <          arch/or32/defconfig          >
arch/or32/defconfig.edited
[shell]# mv arch/or32/defconfig.edited arch/or32/defconfig
[shell]# nice make oldconfig
[shell]# nice make dep
[shell]# nice /usr/bin/make all
```

Après l'installation de uClinux, on est sur d'une chôse: le compilateur gcc pour OpenRISC et Binutilis fonctionnent correctement. Reste à tester le simulateur, cela se fait comme suit:

```
[shell]# touch uart0.rx
[shell]# echo -e "run 40000000 hush\nq" \
| or32-uclinux-sim -f sim.cfg linux -i > /dev/null
```

on obtient alors un fichier nommé uart0.tx dont le contenu doit être similaire ce qui suit:

```
1.
2.
3. OpenRisc 1000 support (C) www.opencores.org
4.
5.
6.
7.
8. uClinux/OR32
9.
```

```
10. Flat model support (C) 1998,1999 Kenneth Albanowski, D.
  Jeff Dionne
11.
12. Calibrating delay loop.. ok - 0.81 BogoMIPS
13.
14. Memory  available:  7904k/8180k  RAM,  0k/0k  ROM  (284k
  kernel data, 450k code)
15.
16. Swansea University Computer Society NET3.035 for Linux
  2.0
17.
18. NET3: Unix domain sockets 0.13 for Linux NET3.035.
19.
20. uClinux                    version               2.0.38.1pre3
  (root@localhost.localdomain)  (gcc  version  3.2.3)  #1  Thu
  Apr 29 11:16:07 CET 2004
21.
22. Serial  driver  version  4.13p1  with  no  serial  options
  enabled
23.
24. ttyS00 at 0x90000000 (irq = 2) is a 16550A
25.
26. Ramdisk driver initialized : 16 ramdisks of 2048K size
27.
28. Blkmem copyright 1998,1999 D. Jeff Dionne
29.
30. Blkmem copyright 1998 Kenneth Albanowski
31.
32. Blkmem 0 disk images:
33.
34. RAMDISK: Romfs filesystem found at block 0
35.
36. RAMDISK: Loading 215 blocks into ram disk... |/-\|/-\|/-
  \|/done.
37.
38. VFS: Mounted root (romfs filesystem).
39.
40. Executing shell ...
41. Shell invoked to run file: /etc/rc
42. Command: ifconfig eth0 inet 10.1.1.133 netmask 255.0.0.0
  hw ether 00:01:02:03:04:05
43. ifconfig: Bad command or file name
44. Command: route add -net 10.0.0.0 netmask 255.0.0.0 dev
  eth0
45. route: Bad command or file name
46. />
```

Cela veut dire que le simulateur a chargé uClinux dans la mémoire du système simulé.

2.14 Installation de uClibc

uClibc est une version allégée de la librairie glibc. Elle est optimisée pour les systèmes embarqués. Un programme compilé en utilisant cette librairie prendra beaucoup moins de place en mémoire.

Pour compiler uClibc, faire ce qui suit:

```
[shell]# ln -s ./extra/Configs/Config.cross.or32.uclinux
Config
[shell]# sed   -e   's/^KERNEL_SOURCE/#KERNEL_SOURCE/;   /
^#KERNEL_SOURCE/ a \
KERNEL_SOURCE   =   ''`pwd`'\/..\/uclinux\/uClinux-2.0.x'   <
Config > Config.edited
[shell]# sed   -e   's/^DEVEL_PREFIX/#DEVEL_PREFIX/;   /
^#DEVEL_PREFIX/ a \
DEVEL_PREFIX = '/opt/or32-uclinux' < Config.edited > Config
[shell]# rm Config.edited
[shell]# nice /usr/bin/make all install
```

Après avoir installé uClibc, il est nécessaire de recompiler gcc afin de prendre en compte cette librairie:

```
[shell]# pushd .
[shell]# cd /opt/or32-uclinux/bin
[shell]# rm -f ar as cc cpp gasp gcc ld nm objcopy objdump
ranlib size strings strip jar grepjar
[shell]# popd
[shell]# cd ../b-gcc
[shell]# nice  env  CC=gcc-295  ../gcc-3.2.3/configure  --
target=or32-uclinux \
--with-gnu-as --with-gnu-ld --verbose \
--enable-threads --prefix=$INSTALL_PREFIX \
--local-prefix=$INSTALL_PREFIX/or32-uclinux \
--enable-languages="c"
[shell]# nice make all install
```

De cette procédure, nous avons développés un script d'installation automatique (cf annexe).

2.15 Évaluation des outils

Lors du développement d'applications pour OpenRISC1200, nous étions face à un problème majeur: les outils de développement sont en cours de développement, ils sont donc parfois instables, buggués et pauvres en documentation sérieuse.

Notre principale source d'information concernant les outils GNU est la documentation GNU officielle de ce outils. Toutefois, cette documentation ne couvre que les aspects généraux ainsi que les informations spécifiques aux plateformes dont le portage est finis, le portage des outils GNU vers OpenRISC1200 n'étant pas achevé, cette documentation ne contient donc pas les informations spécifiques à la plateforme OpenRISC1200.

Nous nous sommes donc basés sur le code source des outils ainsi que sur les forums de discussion et les mailing-lists OpenCores.

Notre souhait au début de ce travail étant de développer des applications à base d'OpenRisc1200, nous nous sommes heurtés aux problèmes relatifs aux outils de développement bas niveau qu'il faut maîtriser. C'est pourquoi nous avons pris la décision de développer des outils haut niveau pour le développement rapide d'applications à base du processeur OpenRISC1200 ce qui est l'objet du chapitre suivant.

Chapitre 3: Conception d'un environnement de développement graphique pour OpenRISC

3.1 Introduction

Lorsque nous avons entamés ce travail, nous avons affrontés un obstacle majeur: Linux est un système d'exploitation qu'on ne peut maîtriser que si l'on maîtrise la ligne de commandes.

En effet, tous les outils de développement GNU sont des outils fonctionnant en mode texte. De plus, la plupart d'entre-eux requièrent d'éditer des fichiers de configuration dont il faut connaître la syntaxe.

Une autre difficulté de taille est la complexité de la procédure d'installation des outils de développement GNU qui ont été portés vers la platefome OpenRISC. Cette difficulté a été atténuée grâce à un script d'installation automatique.

Pour éviter aux débutants dans la plateforme OpenRISC tous ce problèmes afin qu'ils se concentrent dans un premier temps sur les aspects architecturaux, nous avons décidés de développer un environnement graphique pour le développement rapide d'applications à base du processeur OpenRISC. Cette interface constituera notre contribution au projet OpenRISC.

3.2 Objectifs

Les fonctions à implémenter dans la première version de l'interface sont :

- Créer et gérer des projets.
- Paramétrer l'ensemble des outils GNU
- Génération d'un script d'automatisation de la compilation et de génération du fichier de configuration pour le simulateur architectural.

Une fois cette première version finie, nous ambitionnons d'implémenter les fonctions suivantes :

- Syntax highliting (il existe un widget GTK qui effectue cette fonction, il s'agit de scintilla).
- Prise en charge d'Autotools pour la génération automatique des makefiles et des fichiers de préparation à la compilation.
- Connexion directe avec les outils GNU de compilation.
- Connexion directe avec le débogueur GDB.
- Prise en charge du système uClinux.

3.3 Développement

L'interface graphique est développée sous Linux « distribution Fedora 1.0 » et en utilisant les outils suivants:

- GTK+ « the Gimp Tool Kit » version 2.0 [B03]: c'est une librairie C permettant le développement des interfaces graphiques. Elle est sous licence « LGPL » c'est à dire qu'on peut l'utiliser pour développer des logiciels open source et gratuits.
- Gnome [B11]: c'est une librairie graphique basée sur GTK+. C'est la librairie qui est utilisée pour développer le très connu bureau graphique GNOME.
- Glade version 2.0 [A03] c'est une application graphique permettant la création rapide des interfaces graphiques à base des librairies GTK+ et Gnome.
- Anjuta version 1.2.2[T01]: c'est un environnement de développement intégré utilisant le langage C et C++ pour le développement rapide d'applications graphiques GTK+/GNOME. Anjuta peut faire appel à glade pour la création de la partie graphique de notre application, puis il est fait appel à Autotools pour créer l'exécutable c'est à dire l'application finale, le débogage se fait directement à partir de l'environnement graphique en utilisant GDB comme backend.
- Gconf: c'est un système de stockage d'informations de configuration sous la forme « clé=valeur ».
- Devhelp: c'est un outil de documentation et qui a été intégré dans Anjuta.

3.3.1 Présentation de l'interface

L'interface (cf figure) est divisée en trois volets:

- Un gestionnaire de projet à gauche.
- Un éditeur texte à droite.
- Un journal (en bas) qui permet de vois les actions effectuées par le logiciel.

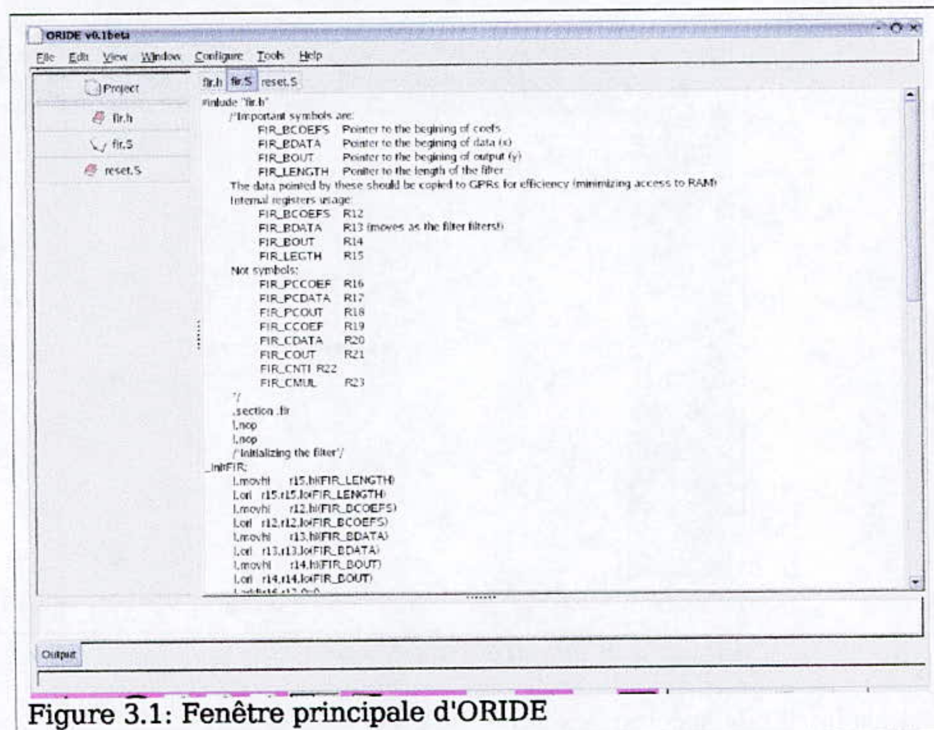


Figure 3.1: Fenêtre principale d'ORIDE

L'interface permet de d'éditer les fichiers source. D'inclure le script du linker (commande configure/link), le Makefile (commande configure/build (Makefile)). Elle contient aussi un assistant pour configurer le simulateur.

Notre logiciel contient aussi un gestionnaire de projets. La création d'un nouveau projet se fait par la commande file/new/project où il faut donner un nom au projet (cf figure).

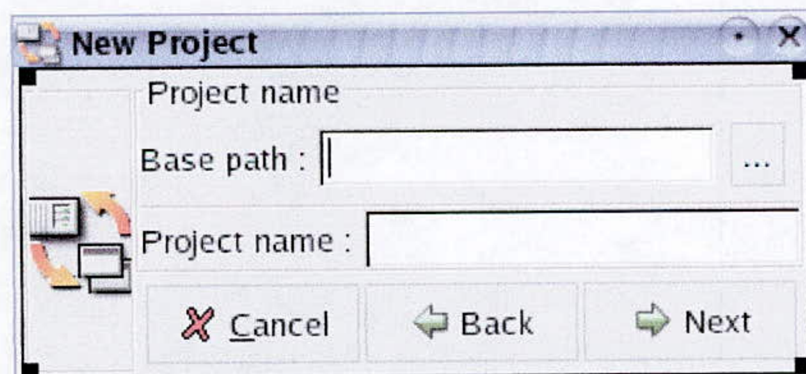


Figure 3.2: Creation d'un nouveau projet

Ensuite, on peut ajouter des fichiers source par la commande file/add/file.

Avant de générer les scripts de compilation et de simulation, il faut exécuter l'assistant configuration du simulateur par la commande configure/simulator. La fenêtre suivante s'affiche alors:

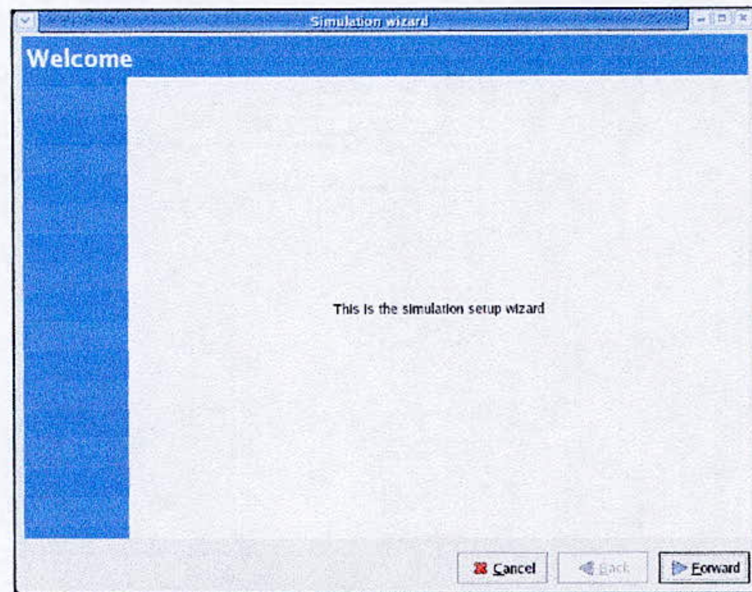


Figure 3.3: Fenêtre d'acceuil de l'assistant

Ensuite, il faut spécifier les unités du processeur à activer :

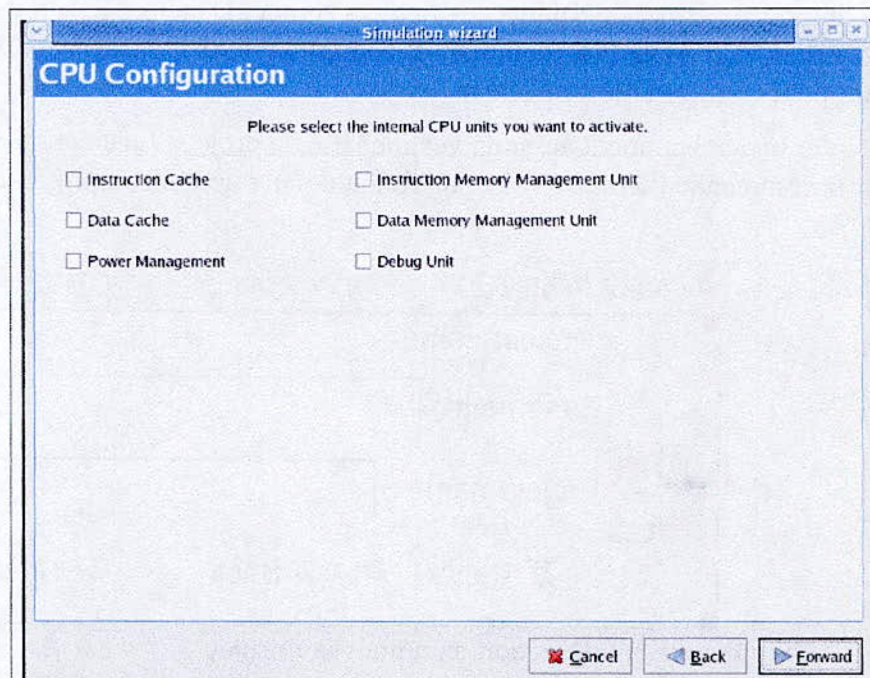


Figure 3.4: Unités du CPU

Si par exemple on choisit cache, lors des étapes suivantes, la fenêtre suivante s'affiche :

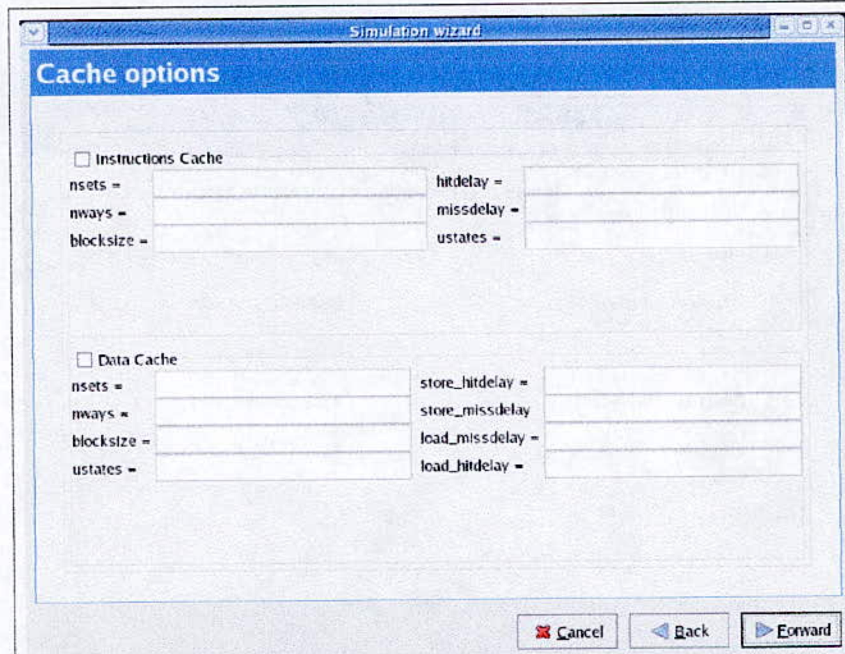


Figure 3.5: Configuration des caches

Par la suite, il faut configurer configurer le CPU :

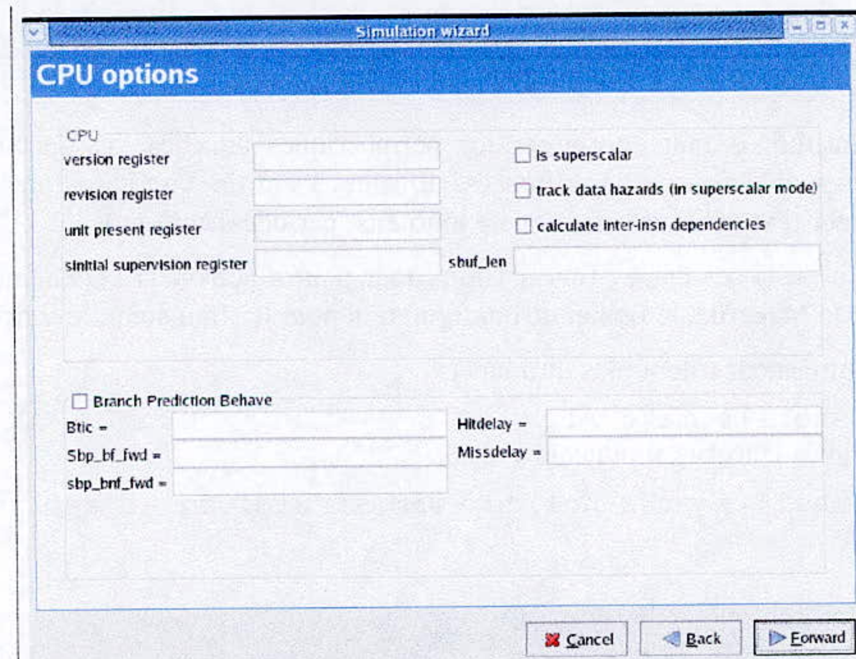


Figure 3.6: Configuration du CPU

Ensuite, on choisit les périphériques à attacher :

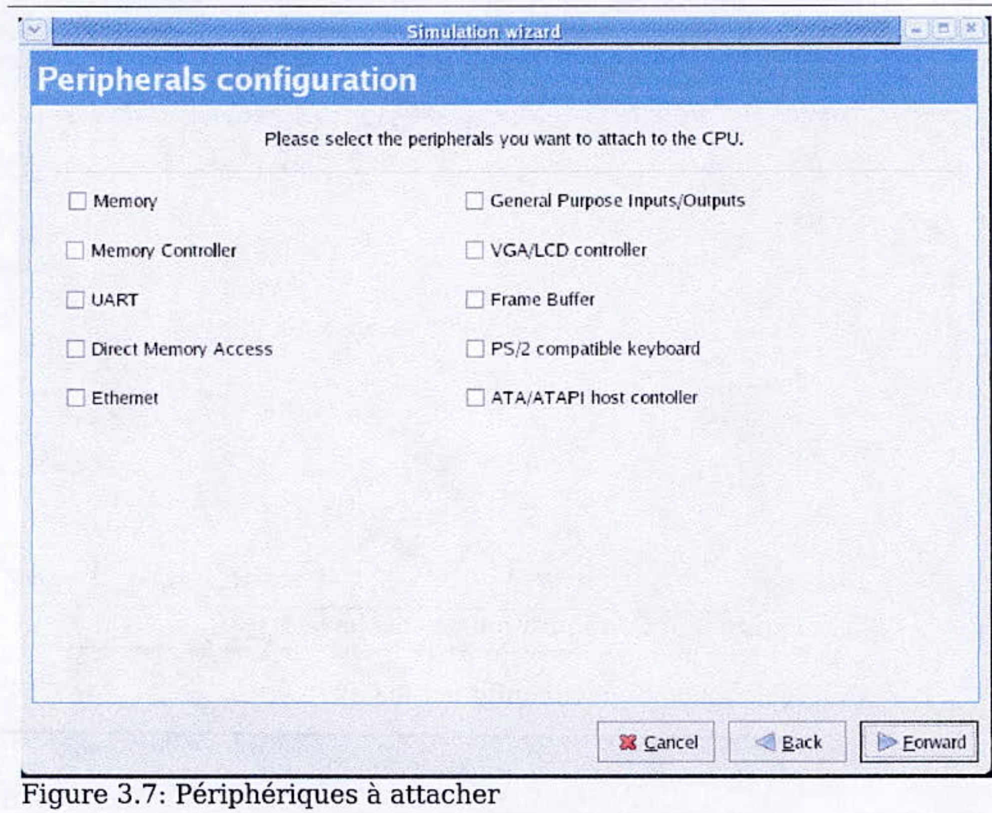


Figure 3.7: Périphériques à attacher

Ensuite, il faut configurer les périphériques attachés, les fenêtres correspondantes à chaque périphérique sont affichées. Ensuite vient la configuration du comportement du simulateur (ex: génération ou non de journaux, période d'horloge).

Une fois ces étapes finies, l'utilisateur pourra activer la comande tools/generate afin de générer le Makefile, le fichier de configuration pour le simulateur, le script linker.

l'utilisateur n'aura plus qu'a taper :

```
[shell]# make all
```

Puis à lancer le simulateur :

```
[shell]# or32-uclinux-sim -f sim.cfg -i application.or32
```

Chapitre 4: Application : Filtre FIR

Dans ce chapitre, nous allons montrer comment utiliser les outils GNU précédemment cités afin de développer des applications à base du processeur OpenRISC.

Nous avons choisis comme application un filtre FIR à 8 coefficients que nous dimensionnerons en utilisant l'outil FDATool (sous Matlab) puis que nous exporterons (avec les données) sous format texte en utilisant l'application genbench que nous avons développés à cet effet.

Ces données seront ensuite liées statiquement à l'exécutable que nous générerons. Le filtre ainsi implémenté n'est donc pas un filtre temps réel. La raison étant que le simulateur architectural ne contenant pas de modèle de simulation pour un système d'acquisition. L'implémentation d'un système temps réel pourra se faire de deux manières (en opérant des modifications mineurs à la solution offline proposée):

- Changer le code source du simulateur architectural (OR1Ksim).
- Transmettre les données en utilisant une uart (le modèle de l'uart est implémenté au niveau du simulateur architectural).

Nous avons choisis d'implémenter un filtre FIR en assembleur.

4.1 Conception du filtre et génération des signaux

Nous utiliserons pour la conception du filtre l'outil FDATool (Filtrer Design and Analysis Tool) fournis avec MATLAB. Cet outil permet de générer un filtre (ie de nous donner les coefficients) à partir du gabarit et des informations sur la structure du filtre à implémenter.

FDATool permet en outre de quantifier toutes les grandeurs manipulées par le filtre, cela en vue de l'implémentation physique car comme nous le savons, les grandeurs dans les circuits numériques sont à précision finies ce qui modifie plus ou moins les performances.

Notre filtre FIR a les spécifications suivantes:

- Filtre passe bas ayant une fréquence de coupure de 9.6 KHz, une ondulation dans la bande passante de 1dB, une atténuation en dehors de la bande passante de 80dB à partir de 12KHz.
- La fréquence d'échantillonnage est de 48KHz.
- La structure du filtre est la forme directe.
- La structure de données, donnée par [n1 n2] où: n1 étant la longueur des mots en bits et n2 la longueur de la partie fractionnelle en bits (la représentation en virgule fixe a

été choisie), est la suivante:

- Les coefficients [16 15].
- Les entrées [16 15].
- Les sorties [32 30].
- Les multiplicandes [16 15].
- Les produits [32 30].
- Les sommes [32 30].

On lance FDATool en tapant la commande ***fdatool*** dans l'invite MATAB, la fenêtre suivant s'affiche alors:

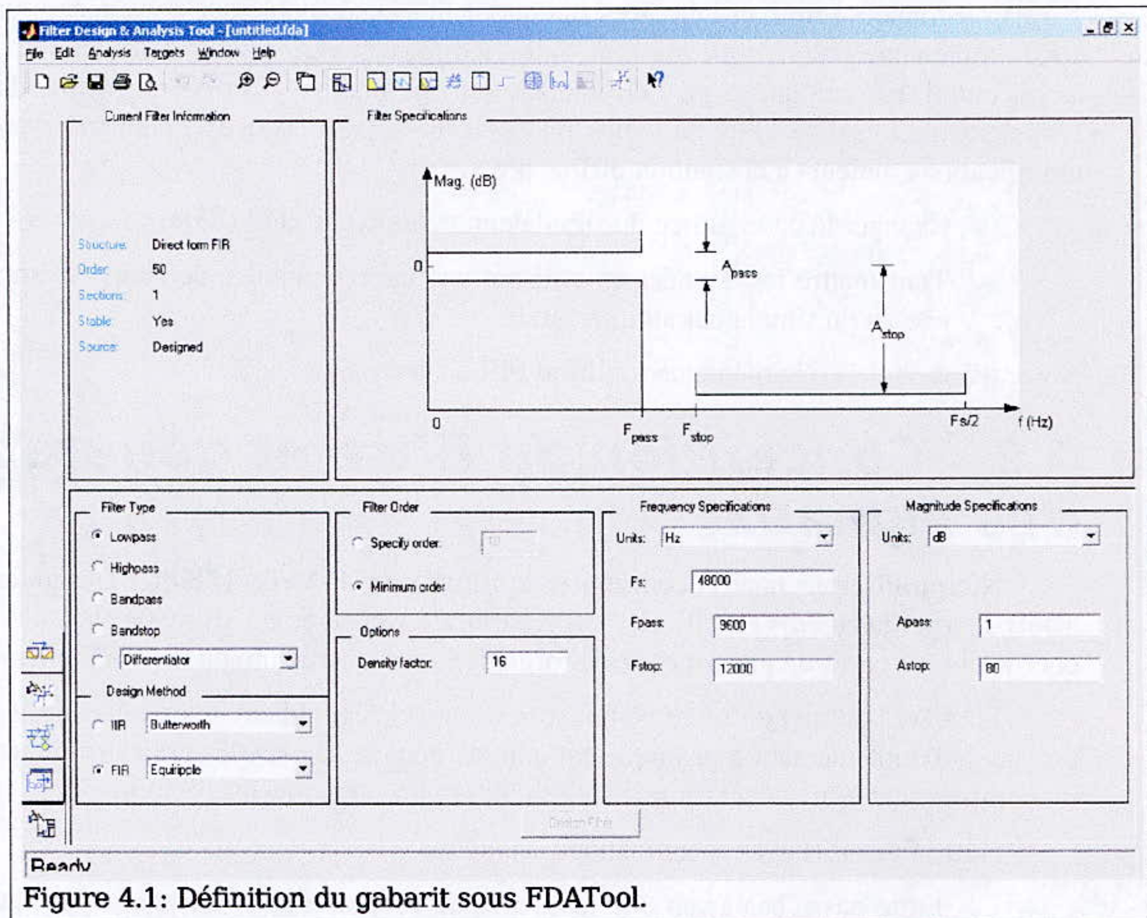


Figure 4.1: Définition du gabarit sous FDATool.

En définissant les caractéristiques du filtre passe bas, la fenêtre suivante s'affiche:

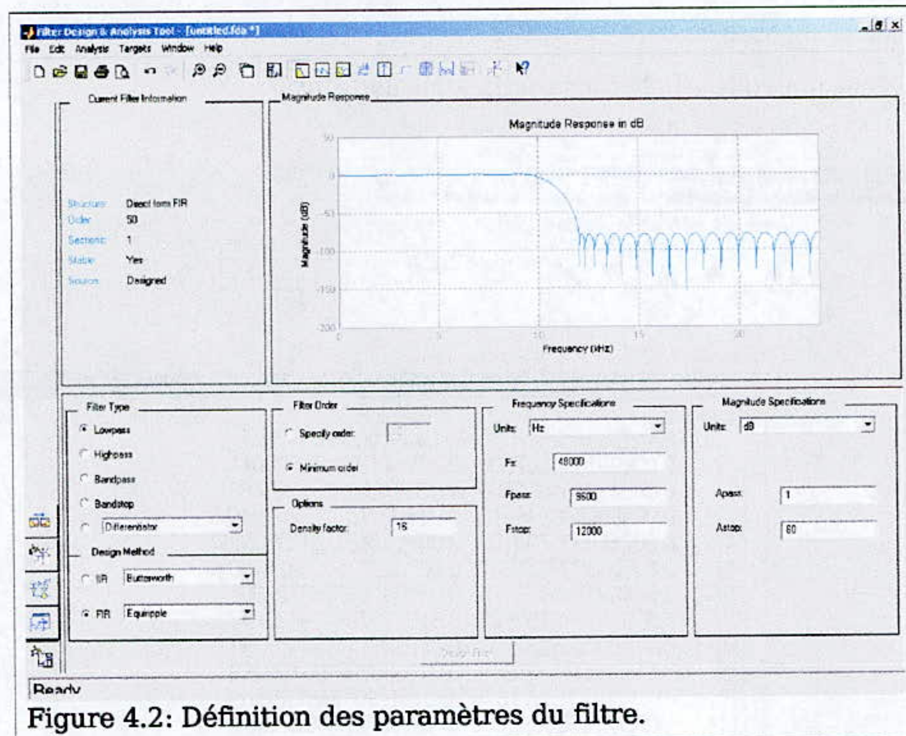


Figure 4.2: Définition des paramètres du filtre.

Pour définir les paramètres de quantification, nous devons activer la quantification, la fenêtre suivante s'affiche alors:

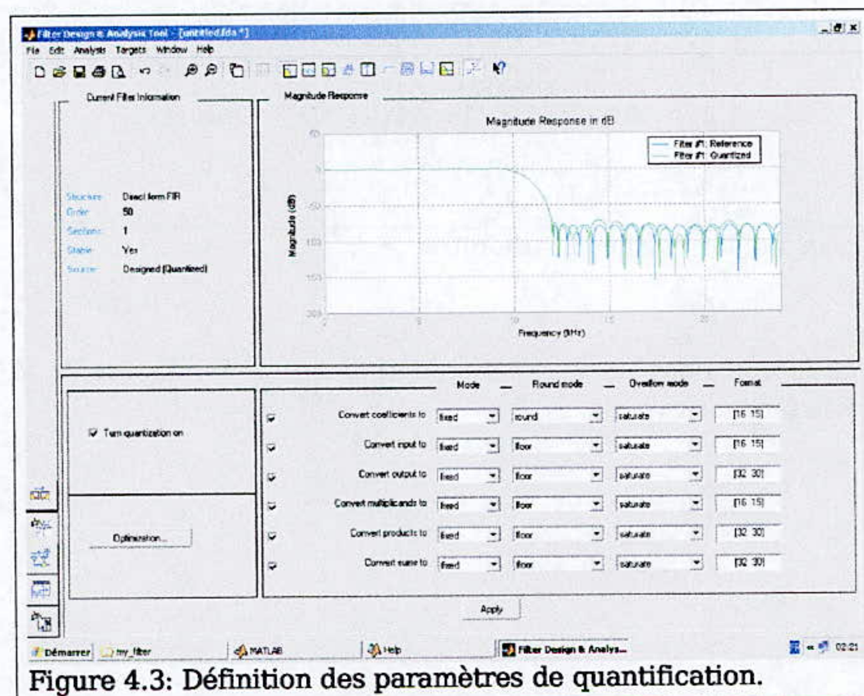
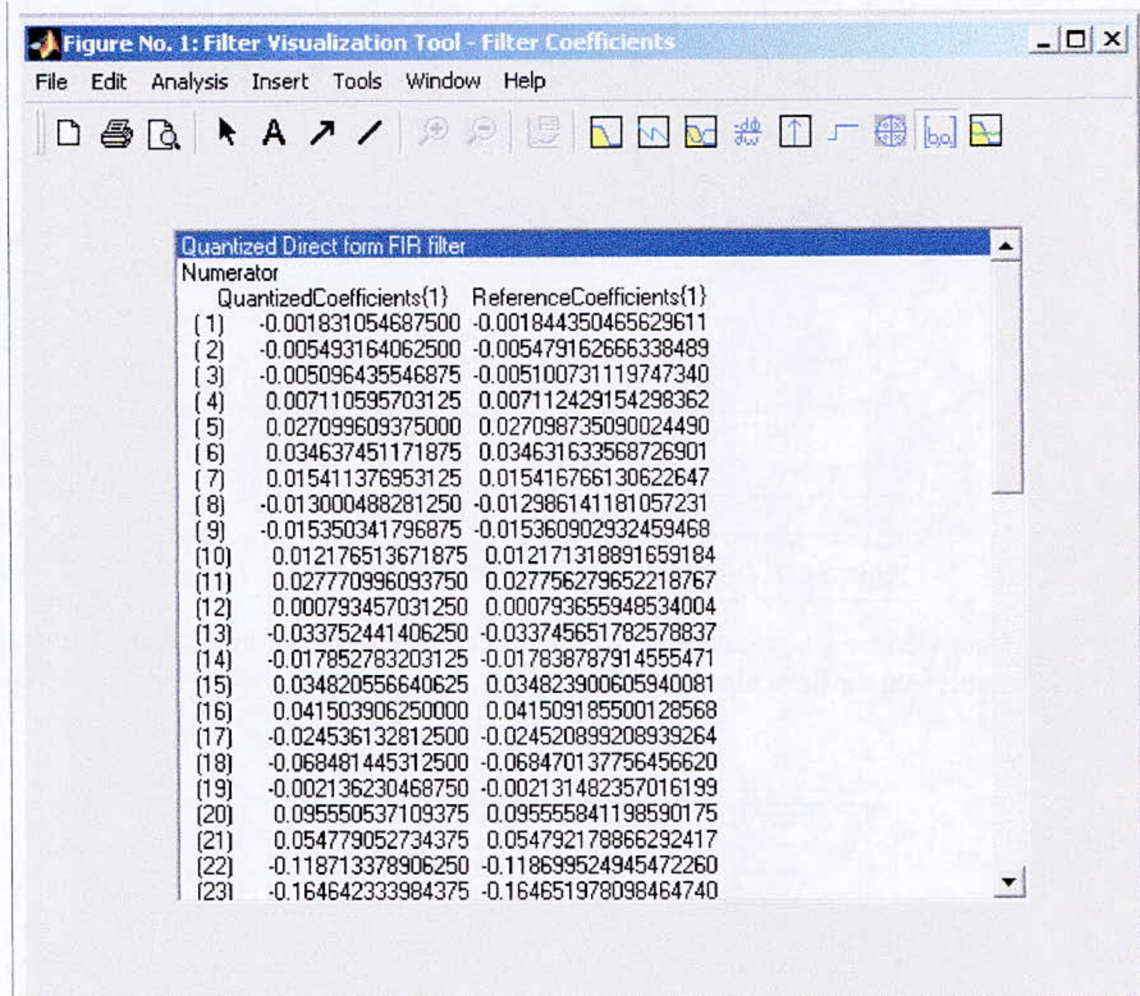


Figure 4.3: Définition des paramètres de quantification.

Après activation de la quantification, nous voyons la réponse impulsionnelle théorique

ainsi que la réponse du filtre quantifié.

Nous pouvons afficher les coefficients du filtre:



	QuantizedCoefficients{1}	ReferenceCoefficients{1}
(1)	-0.001831054687500	-0.001844350465629611
(2)	-0.005493164062500	-0.005479162666338489
(3)	-0.005096435546875	-0.005100731119747340
(4)	0.007110595703125	0.007112429154298362
(5)	0.027099609375000	0.027098735090024490
(6)	0.034637451171875	0.034631633568726901
(7)	0.015411376953125	0.015416766130622647
(8)	-0.013000488281250	-0.012986141181057231
(9)	-0.015350341796875	-0.015360902932459468
(10)	0.012176513671875	0.012171318891659184
(11)	0.027770996093750	0.027756279652218767
(12)	0.000793457031250	0.000793655948534004
(13)	-0.033752441406250	-0.033745651782578837
(14)	-0.017852783203125	-0.017838787914555471
(15)	0.034820556640625	0.034823900605940081
(16)	0.041503906250000	0.041509185500128568
(17)	-0.024536132812500	-0.024520899208939264
(18)	-0.068481445312500	-0.068470137756456620
(19)	-0.002136230468750	-0.002131482357016199
(20)	0.095550537109375	0.095555841198590175
(21)	0.054779052734375	0.054792178866292417
(22)	-0.118713378906250	-0.118699524945472260
(23)	-0.164642333984375	-0.164651978098464740

Figure 4.4: les coefficients du filtre.

Par la suite nous allons exporter le filtre dans le Workspace de MATLAB, cela se fait en utilisant la fonction export:

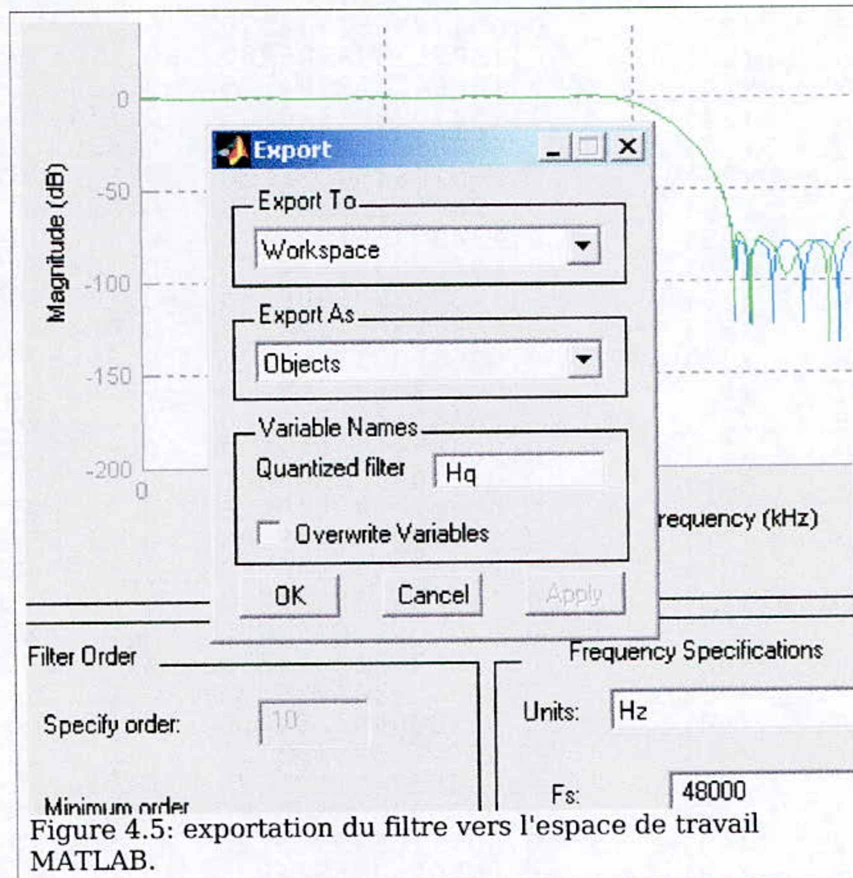


Figure 4.5: exportation du filtre vers l'espace de travail MATLAB.

Si maintenant on tape My_fir sous matlab, nous obtenons:

```
1. My_fir =
2. Quantized Direct form FIR filter
3. Numerator
4.            QuantizedCoefficients{1}     ReferenceCoefficients
   {1}
5.     ( 1)        -0.001831054687500  -0.001844350465629611
6.     ( 2)        -0.005493164062500  -0.005479162666338489
7.     ( 3)        -0.005096435546875  -0.005100731119747340
8.     ( 4)         0.007110595703125   0.007112429154298362
9.     ( 5)         0.027099609375000   0.027098735090024490
10.     ( 6)         0.034637451171875   0.034631633568726901
11.     ( 7)         0.015411376953125   0.015416766130622647
12.     ( 8)        -0.013000488281250  -0.012986141181057231
13.     ( 9)        -0.015350341796875  -0.015360902932459468
14.     (10)         0.012176513671875   0.012171318891659184
15.     (11)         0.027770996093750   0.027756279652218767
16.     (12)         0.000793457031250   0.000793655948534004
17.     (13)        -0.033752441406250  -0.033745651782578837
18.     (14)        -0.017852783203125  -0.017838787914555471
19.     (15)         0.034820556640625   0.034823900605940081
20.     (16)         0.041503906250000   0.041509185500128568
21.     (17)        -0.024536132812500  -0.024520899208939264
22.     (18)        -0.068481445312500  -0.068470137756456620
23.     (19)        -0.002136230468750  -0.002131482357016199
24.     (20)         0.095550537109375   0.095555841198590175
```

```
25.    (21)       0.054779052734375   0.0547921788662924l7
26.    (22)      -0.118713378906250  -0.118699524945472260
27.    (23)      -0.164642333984375  -0.164651978098464740
28.    (24)       0.134307861328125   0.134304440902682760
29.    (25)       0.620056152343750   0.620046958666089500
30.    (26)       0.860168457031250   0.860175170356427100
31.    (27)       0.620056152343750   0.620046958666089500
32.    (28)       0.134307861328125   0.134304440902682760
33.    (29)      -0.164642333984375  -0.164651978098464740
34.    (30)      -0.118713378906250  -0.118699524945472260
35.    (31)       0.054779052734375   0.054792178866292417
36.    (32)       0.095550537109375   0.095555841198590175
37.    (33)      -0.002136230468750  -0.002131482357016199
38.    (34)      -0.068481445312500  -0.068470137756456620
39.    (35)      -0.024536132812500  -0.024520899208939264
40.    (36)       0.041503906250000   0.041509185500128568
41.    (37)       0.034820556640625   0.034823900605940081
42.    (38)      -0.017852783203125  -0.017838787914555471
43.    (39)      -0.033752441406250  -0.033745651782578837
44.    (40)       0.000793457031250   0.000793655948534004
45.    (41)       0.027770996093750   0.027756279652218767
46.    (42)       0.012176513671875   0.012171318891659184
47.    (43)      -0.015350341796875  -0.015360902932459468
48.    (44)      -0.013000488281250  -0.012986141181057231
49.    (45)       0.015411376953125   0.015416766130622647
50.    (46)       0.034637451171875   0.034631633568726901
51.    (47)       0.027099609375000   0.027098735090024490
52.    (48)       0.007110595703125   0.007112429154298362
53.    (49)      -0.005096435546875  -0.005100731119747340
54.    (50)      -0.005493164062500  -0.005479162666338489
55.    (51)      -0.001831054687500  -0.001844350465629611
56.
57.     FilterStructure = fir
58.         ScaleValues = [0.5]
59.    NumberOfSections = 1
60.    StatesPerSection = [50]
61.     CoefficientFormat  =  quantizer('fixed',  'round',
  'saturate', [16  15])
62.           InputFormat  = quantizer('fixed',  'floor',
  'saturate', [16  15])
63.          OutputFormat  =  quantizer('fixed',  'floor',
  'saturate', [32  30])
64. MultiplicandFormat   =   quantizer('fixed',   'floor',
  'saturate', [16  15])
65.         ProductFormat  =  quantizer('fixed',  'floor',
  'saturate', [32  30])
1.              SumFormat  =  quantizer('fixed',  'floor',
  'saturate', [32  30])
```

4.2 Génèration des stimulus

Pour générer les stimulus, nous utilisons les filtres de quantification contenus dans la structure du filtre précédemment exporté vers MATLAB.

Pour faciliter cette tâche, nous avons développés un logiciel fonctionnant sous MATLAB

Nous avons baptisés ce logiciel « GenBench ».

Pour le lancer, il suffit de taper « genbench » dans l'invite MATLAB, nous obtenons alors la fenêtre principale:

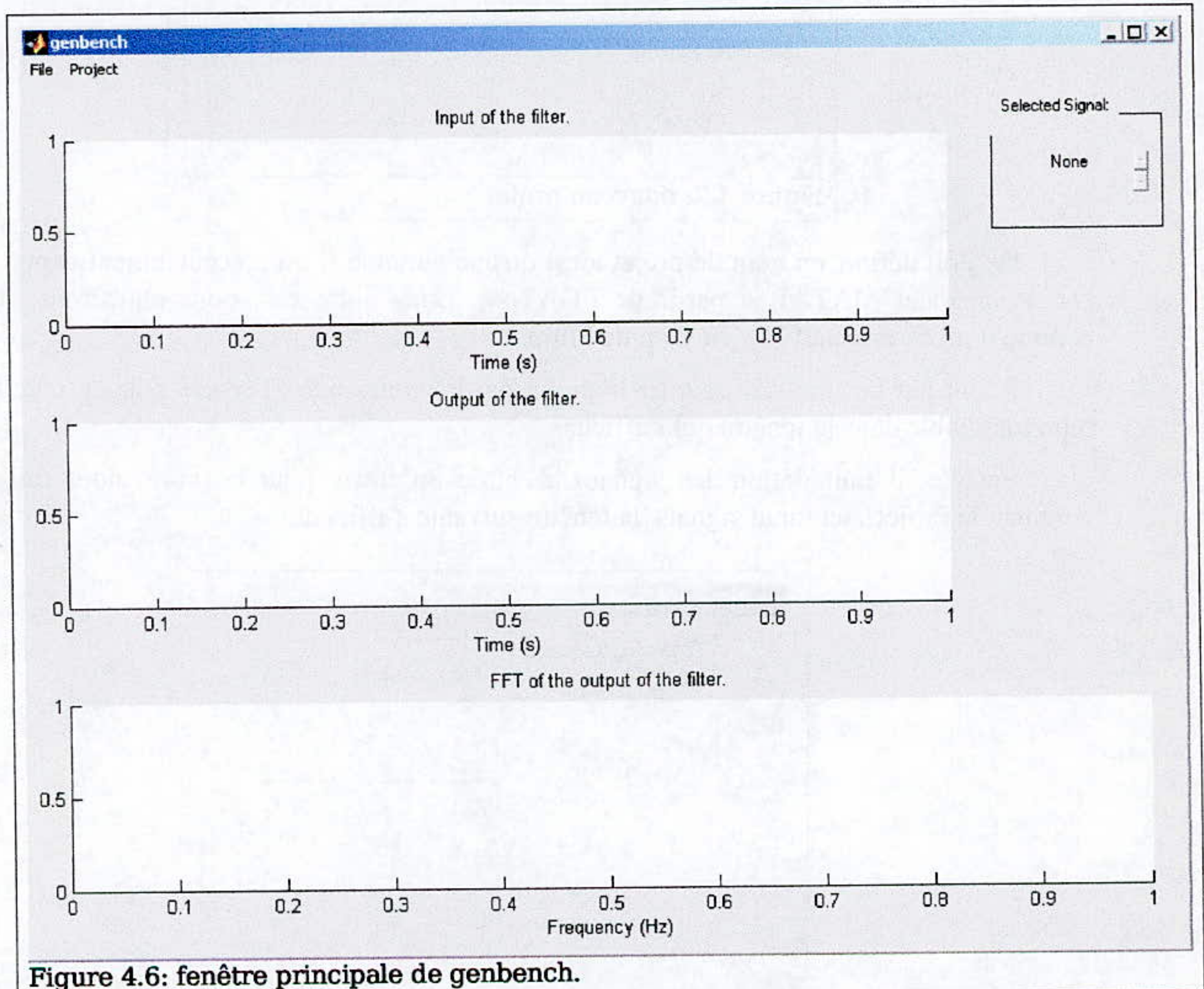


Figure 4.6: fenêtre principale de genbench.

Nous préférons donner le fonctionnement de ce logiciel par l'exemple suivant:

Tout d'abord, nous allons créer un projet, cela se fait en cliquant sur le menu File/New, nous obtenons la figure suivante:

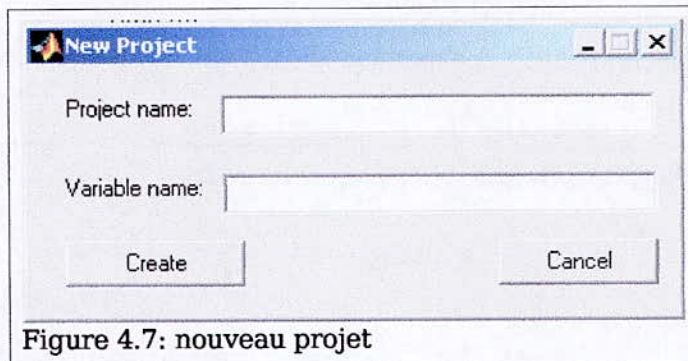


Figure 4.7: nouveau projet

On doit définir un nom de projet ainsi qu'une variable filtre précédemment exportée dans l'environnement MATAB à partir de FDATool. Dans notre cas, nous choisirons my_filter comme nom de projet et My_fir comme Filtre.

Il faut par la suite sauvegarder le projet par la commande File/save puis en spécifiant un répertoire cible dans la fenêtre qui s'affiche.

Ensuite, il faut définir les signaux d'entrée du filtre, pour ce faire, nous utilisons la commande Project/set input signals, la fenêtre suivante s'affiche:

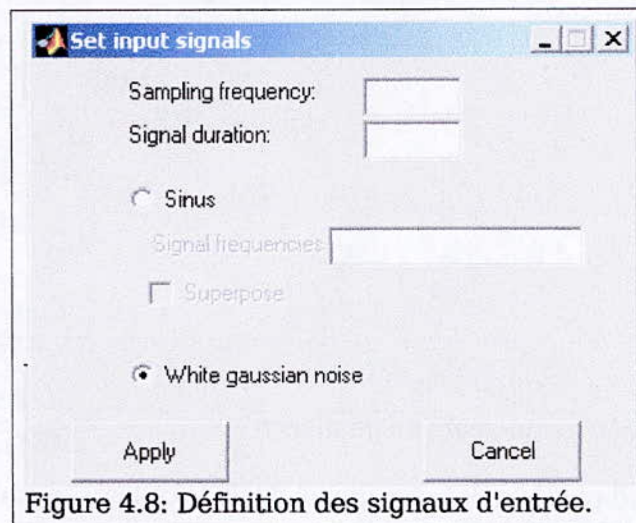


Figure 4.8: Définition des signaux d'entrée.

Il est possible de définir deux classes de signaux, soit des signaux sinusoïdaux avec possibilité de superposition ou alors un bruit blanc gaussien.

Pour notre exemple, nous allons choisir un bruit blanc gaussien avec une fréquence d'échantillonnage de 48KHz qui doit être égale à celle utilisée lors de la conception du filtre. La durée du signal sera de 0.05s.

L'étape suivante consiste à générer les stimulus (le fichier contenant les coefficients et les signaux au format binaire). Pour ce faire, on utilise la commande Project/generate testbench data. Pendant ce processus, la fenêtre suivante s'affiche:

A ce stade, plusieurs fichiers ont été générés dans le répertoire choisis précédemment:

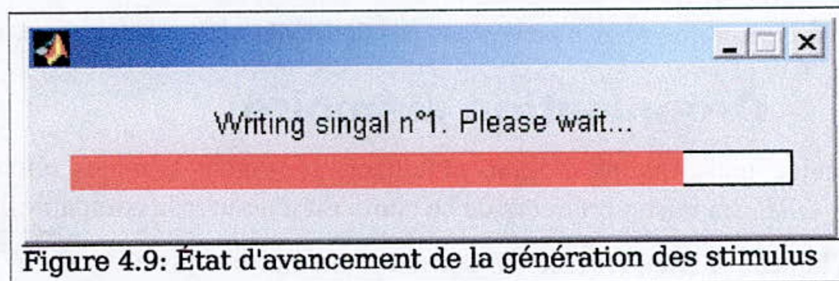


Figure 4.9: État d'avancement de la génération des stimulus

- Un fichier projet, dans notre cas « my_filter.prj ».
- Un fichier contenant les coefficients du filtre, dans notre cas « my_filter.coe ».
- Un fichier contenant les signaux générés, dans notre cas « my_filter.sig ».

4.3 Architecture choisie

Pour notre application, nous nous baserons sur la spécification ORP (OpenRisc Reference Platform) qui définit les plages de mémoire, les interruptions ainsi que les cannaux DMA à utiliser pour différents périphériques lors de la conception d'un système à base d'OpenRISC.

L'architecture choisie est composée d'une mémoire Flash, d'une mémoire RAM, d'une UART, d'un contrôleur de mémoire [B09]et d'un processeur OpenRISC.

La figure suivante illustre un tel système:

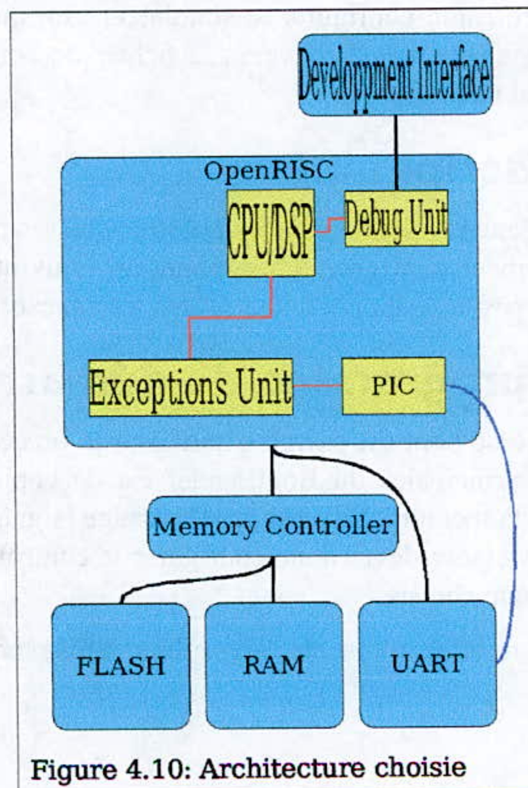


Figure 4.10: Architecture choisie

Il est à noter que nous n'activerons pas de MMU ni de mémoire cache.

4.3.1 Organisation mémoire

Comme nous l'avons indiqué plus haut, nous nous sommes référés à la plateforme de référence ORP. La raison première de ce choix est d'assurer la compatibilité des programmes.

Le tableau suivant résume l'organisation mémoire choisie :

Désignation	*Adresse de base*	*Taille*	*Adresse lors du boot*
Mémoire Flash	0xF0000000	0x00800000	0x0
Mémoire RAM1	0x00000000	0x00800000	N/A
Mémoire RAM2	0x00800000	0x00400000	N/A
Contrôleur mémoire	0x93000000	0x00000050	0x93000000
UART	0x90000000		N/A

Table 4.1: Memory map

4.3.2 Configuration du simulateur

Il faut, bien entendu, configurer le simulateur afin qu'il prenne en compte l'architecture choisie. Dans les annexes, vous trouverez le fichier de configuration du simulateur (sim.cfg) généré par le logiciel ORIDE.

4.3.3 Le script Linker

Afin que l'exécutable pointe vers les bonnes adresses mémoire, il est indispensable que le linker soit correctement configuré. C'est pourquoi nous utilisons un script de configuration personnalisé et non pas le script par défaut. Dans les annexes se trouve le script du linker.

4.3.4 Le programme de démarrage (boot)

Le programme de boot est chargé d'initialiser le processeur et les différents périphérique. Une des fonctions principales du BootLoader est de copier le vecteur des exceptions de la mémoire flash vers la mémoire vive car au démarrage (signal reset), la flash démarre à l'adresse 0x00000000, le processeur devra donc configurer le contrôleur de mémoire afin de prendre en charge le memory map choisis.

L'organigramme suivant résume l'algorithme utilisé lors du démarrage:

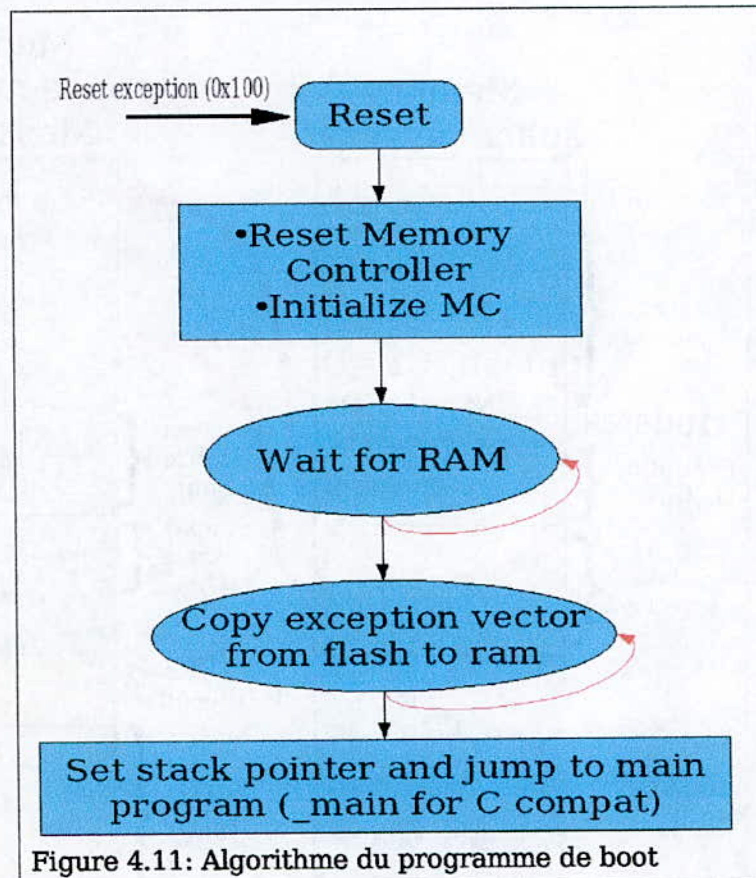


Figure 4.11: Algorithme du programme de boot

Le programme de boot sera chargé dès la réception d'une exception reset, cette exception est reçue par l'intermédiaire du signal reset externe qui est généré par la circuiterie d'initialisation de la carte. Cette circuiterie est aussi responsable de l'initialisation du registre POC (Power On Configuration) du contrôleur de mémoire afin que la mémoire flash puisse être correctement lue. La figure suivante illustre les changements au niveau du memory map entre le moment où le reset survient et le moment ou la procédure de boot prend fin:

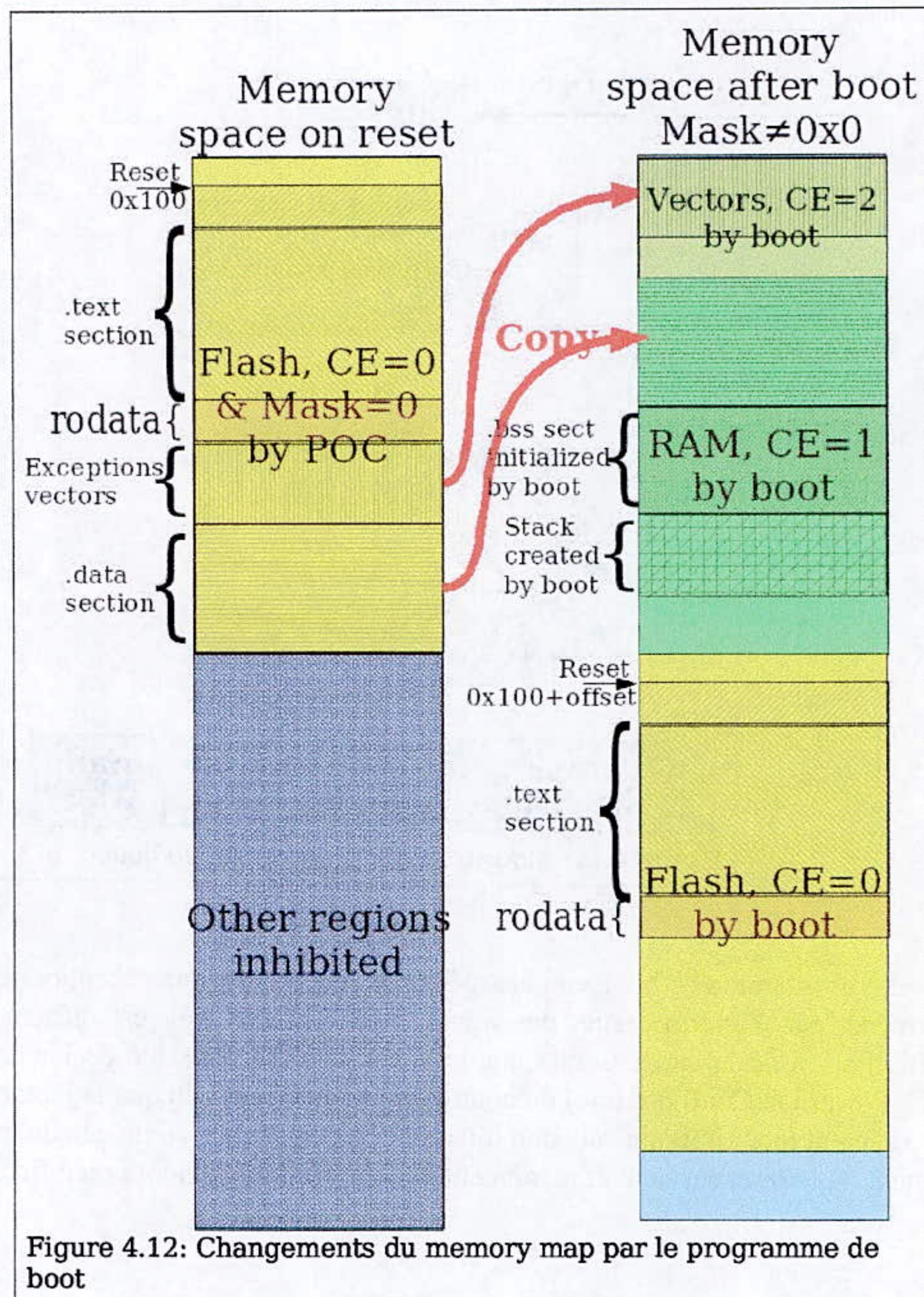


Figure 4.12: Changements du memory map par le programme de boot

Le rôle du bootloader est donc de préparer la mémoire afin que les programmes y trouvent les données dont ils ont besoin. Dans le cas de notre filtre FIR, les coefficiens ainsi que les données devront se situer dans les sections: .coefs et .input_sig dans la mémoire flash.

4.4 Développement Filtre FIR

Après configuration de la mémoire par le bootloader, le processeur passe en mode utilisateur et le programme principal est lancé.

Dans notre cas le programme principal effectue un filtrage FIR.

4.4.1 Les symboles utilisés

Les symboles utilisés par le programme de filtrage sont donnés par la liste suivante, leur valeur est définie au moment de l'édition des liens puisqu'ils sont définis dans le script de l'éditeur des liens:

FIR_BCOEFS	Pointeur vers le début des coefficient
FIR_BDATA	Pointeur vers le début des données (x)
FIR_BOUT	Pointeur vers le début de la sortie (y)
FIR_LENGTH	Pointeur vers la taille du filtre (N)
FIR_LENGTHX	Pointeur vers la taille des données (x(nmax))

D'autres symboles sont utilisés par le programme, ces symboles sont définis dans les différents fichiers d'en-tête qui sont fournis avec les outils de développement. Ces fichiers d'en-tête contiennent des symboles qui définissent les registres internes du processeur ainsi que des périphériques.

4.4.2 Utilisation des registres internes

La liste suivante résume le rôle que joue chaque registre interne dans l'opération de filtrage:

Pointeur vers le début des coefs	R12
Pointeur vers le début des données	R13 (change lors du filtrage)
Pointeur vers la dernière donnée à filtrer	R24
Pointeur vers le début des sorties	R14
Longueur du filtre	R15
Pointeur vers le coeficient actuel	R16
Pointeur vers la donnée actuelle	R17
Pointeur vers y actuel	R18
Pointeur vers le coef actuel	R19
Donnée actuelle	R20
Sortie actuelle	R21
Compteur (i dans l'algorithme du filtre)	R22
Résultat de la multiplication	R23

4.4.3 Optimisations

Lors de l'implémentation de la version assembleur du filtre, une attention particulière à été donnée quand à l'exploitation des avantages de l'architecture Harvard ainsi que des « delay

slots » (qui sont des instructions placées après les branchements et qui s'exécutent toujours) lors des branchements. Se réferrer au code source du programme dans les annexes.

4.4.4 Utilisation d'ORIDE

Il suffit, pour développer le filtre, d'ajouter un projet (fir par exemple).

Il faut ensuite éditer les fichiers:

- fir.S : fichier assembleur contenant l'algorithme de filtrage.
- reset.S : fichier assembleur contenant le programme de démarrage (boot).
- Data.S : fichier assembleur qui contient le signal à filter (placé dans la section . input_sig) ainsi que les coefficients (placés dans la section .coefs) du filtre.
- spr_defs.h : fichier en-tête contenant la définition des symboles représentant les adresses des registres à usage spécial.
- mc.h : définition des registres du contrôleur mémoire.
- board.h : définition de la carte.

Il est indispensable de mettre à jour les fichiers .h afin qu'ils reflètent la configuration de la carte et en particulier le memory map.

Ensuite il faut configurer le Makefile (configure/build) en ajoutant le contenu de l'annexe.

Aussi configurer le linker (configure/link) en ajoutant le contenu de l'annexe.

Pour récupérer les signaux générés avec Matlab (genbench), nous avons écris un petit programme (en Perl) qui récupère ces données et les place dans un fichier assembleur. Pour générer un fichier *data.S* (assembleur qui contiendra les données et les coefficients) à partir des fichiers *My_filter.sig* et *My_filter.coe*, il faut taper le commande:

```
[shell]# ./data2asm.pl My_filter.coe My_filter.sig data.S
```

Le programme affichera :

```
1. dos2unix: converting file My_fiter.coe to UNIX format ...
2. dos2unix: converting file My_fiter.sig to UNIX format ...
3.
4. Done...
```

Il est à noter que cette étape a été incluse dans le *Makefile.*

Le fichier data.S résultant de cette conversion est donné dans l'annexe.

4.4.5 Compilation

Pour compiler, il suffira de taper à partir du shell:

```
[shell]# make all
```

Le fichier exécutable généré sera *My_filter.or32*, d'autres fichiers très utiles sont aussi générés: *hexa.txt* qui contiendra le fichier exécutable mais présenté sous format hexadécimal en montrant la limite des différentes sections, le fichier *disass.txt* qui contient le programme désassemblé et le fichier *syms.txt* qui contient la liste des symboles.

4.4.6 Simulation et validation des résultats

Pour lancer le simulateur, il suffit de taper :

```
[shell]# or32-uclinux-sim -i -f sim.cfg My_filter.or32
1. Reading script file from 'sim.cfg'...
2. Verbose on, simdebug off, interactive prompt on
3. Machine initialization...
4. Clock cycle: 4ns
5. No data cache.
6. No instruction cache.
7. BPB simulation off.
8. BTIC simulation off.
9.
10. Building automata... done, num uncovered: 0/212.
11. Parsing operands data... done.
12. loadcode:    filename    my_filter.or32           startaddr=0
  virtphy_transl=0
13. Not COFF file format
14. ELF type: 0x0002
15. ELF machine: 0x8472
16. ELF version: 0x00000001
17. ELF sec = 15
18. Section: .reset, vaddr: 0xf0000000, paddr: 0xf0000000,
  offset: 0x00002000, size: 0x000001a0
19. Section: .text, vaddr: 0xf00001a0, paddr: 0xf00001a0,
  offset: 0x000021a0, size: 0x000001dc
20. Section: .coefs, vaddr: 0xf000037c, paddr: 0xf000037c,
  offset: 0x0000237c, size: 0x00000066
21. Section:   .input_sig,   vaddr:   0xf00003e4,   paddr:
  0xf00003e4, offset: 0x000023e4, size: 0x000012c2
22. Section: .data, vaddr: 0x00002000, paddr: 0x00002000,
  offset: 0x00004000, size: 0x00000000
23. Resetting Tick Timer.
24. Resetting Power Management.
25. Resetting PIC.
26. Resetting memory controller.
27. Starting at 0x00000000
28. Exception 0x100 (Reset) at 0x0, EA: 0x0, ppc: 0x0, npc:
  0x4, #0
29. (sim)
```

On peut alors définir un point d'arrêt, à la fin du programme qui correspond à l'étiquette _dump_mem qui a pour valeur (en se basant sur syms.txt : table des symboles) 0xF0000370:

```
(sim)      break 0xf0000370
```

Puis, on lance le simulateur, le chiffre représente le nombre d'instructions à exécuter.

```
(sim)      Run 1342560
  1. ....
  2. Breakpoint hit.
```

Le simulateur stoppe, nous pouvons alors envoyer le résultat du filtrage dans un fichier par la commande ***dm*** (display memory) adresse_début adresse_fin > *memdump.txt*

Ensuite, il suffit de récupérer le contenu de ce fichier (qui est codé en hexadécimal), de le

mettre en forme par le programme ***dump2wintxt*** puis de la charger dans le logiciel genbench (commande project/load results).

Nous obtenons alors les graphes suivants (la premier signal étant l'entrée, le second la sortie et le troisième le spectre de la sortie) :

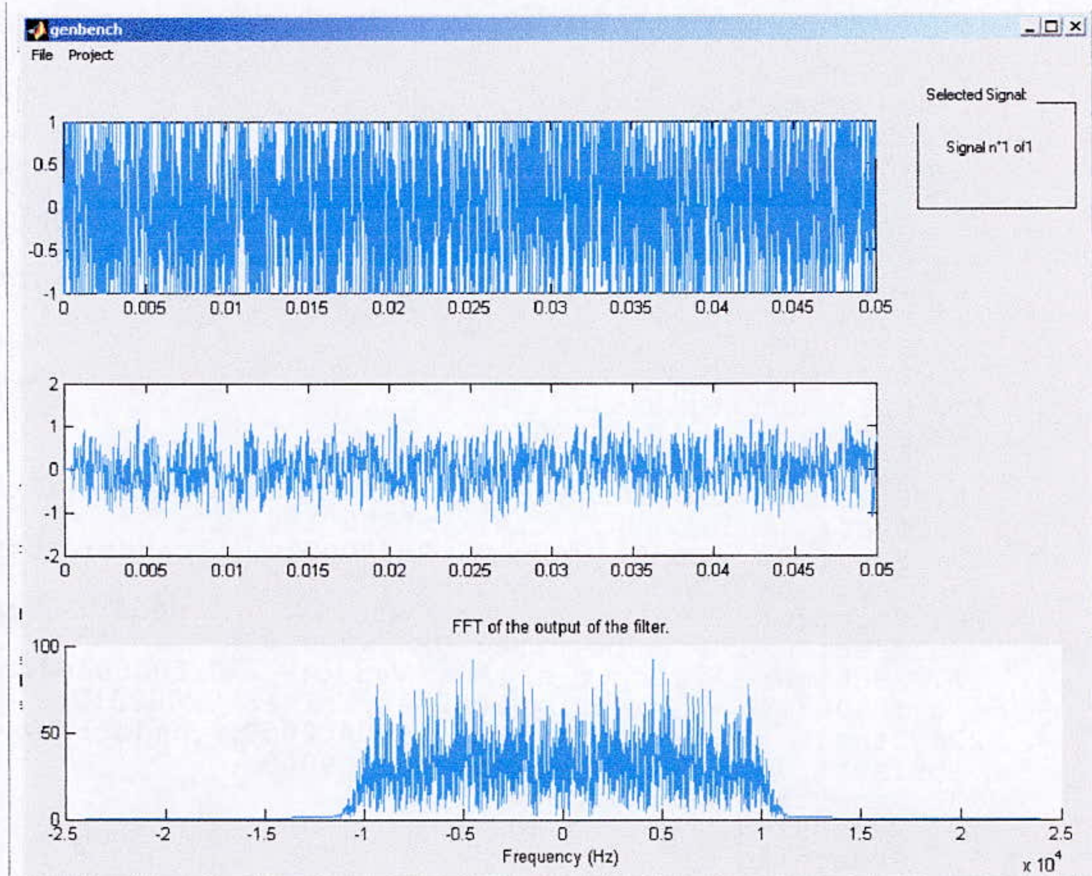


Figure 4.13: visualisation des résultats.

4.4.7 Conclusions

Il est claire que l'implémentation d'un algorithme au niveau assembleur est très compliquée car il faut prendre en compte tous les aspects relatifs au processeur et à son entourage (mémoires et périphériques). C'est pourquoi le développement des programmes est relativement long.

Toutefois, nous avons la possibilité de contrôler le fonctionnement « intime » du système et ainsi d'optimiser au maximum le déroulement de l'algorithme.

Conclusions et perspectives

Conclusions

Ce PFE nous a permis d'atteindre un certain nombre d'objectifs : le plus important est la familiarisation avec le développement d'applications pour les systèmes à base du processeur OpenRISC.

Nous avons aussi eu l'occasion de constater que les outils OpenSource pouvaient être aussi performants que leurs équivalents propriétaires si leur développement est rigoureusement géré. Les outils OpenSource nous ont permis aussi d'élargir nos connaissance dans le domaine du développement à une vitesse qui aurait été impossible si on utilisait des outils proprétaires, cela est surtout dû à l'esprit de partage et de collaboration qu'anime la comunauté OpenSource.

Le choix d'utiliser des outils OpenSource se révèle donc une très bonne solution alternative pour l'accès aux technologies modernes surtout dans l'environnement économique de notre pays, pour peu qu'il y ait les compétences et la volonté d'en tirer le maximum.

Perspectives

Il est clair que ce travail n'en est qu'à son début. En effet, nous ambitionnons de faire évoluer le logiciel ORIDE (avec l'aide de la communauté de développeurs OpenSource) vers une solution de développement complète à base du core OpenRISC1000.

L'idéal serait d'avoir une partie de développement Hardware basée sur IVI (Interactive Verilog Icarus) qui est un environnement de développement qui utilise le language de description materiel (HDL Hardware Description Language). Et aussi une partie de développement software similaire aux outils TexasInstruments par exemple.

La disponibilité de tels outils est indispensable afin que le développement d'applications à base d'OpenRISC soit réellement rentable car il ne faudrait pas perdre en temps ce que l'on gagne sur le prix du core. Et ainsi l'utilisation d'OpenRisc deviendra sûrement un concurrent sérieux des cores propriétaires.

Références Bibliographiques

Les Références bibliographiques peuvent être :

BXX : Livre.

AXX : Article

TXX : Documentation technique (par exemple le manuel en ligne d'un logiciel).

WXX : site web.

A03: Eddy Ahmed, Developing Gnome Apps with Glade, 2004
A04: {Bolado,Posadas,Castillo,Huerta,P.Sanchez}(1), {C.Shanchez,Fouren,Blasco}(2), Platformbased on Open-Source Cores for Industrial Applications, 2004
B01: Stephen Figgins, Ellen Siever, Aaron Weber, Linux in a Nutsell, 4th Edition, 2003
B03: Tony Gale, GTK+ 2.0 Tutorial, 2003
B04: Damjan LAmpret, Maria Bolado, OpenRisc 1000 Architecture Manual, 2003
B05: Red Hat, Inc, Red Hat Entreprise Linux 3 "Developer Tools Guide", 2003
B06: Red Hat, Inc, Red Hat Entreprise Linux 3 "Using binutils, the Gnu Binary utilities", 2003
B07: Red Hat, Inc, Red Hat Entreprise Linux 3 "Using ld, the Gnu Linker", 2003
B08: Red Hat, Inc, Red Hat Enterprise Linux 3Using as, the Gnu Assembler, 2003
B09: Rudolf Usselmann, Memory Controller IP Core, 2002
B10: Damjan Lampret, OpenRISC 1200IP CoreSpecification (rev 0.6), 2001
T01: Olivier Pincon, Naba Kumar, Andy Piper, Anjuta documentation, 2002
W05: Damjan Lampret, OpenRISC 1000 Overview, 2004, www.opencores.com/projects.cgi/web/or1k/overview
W06: Marko Mlinar, Damjan Lampret., OpenRISC 1000: GNU Toolchain Port, 2004, opencores.com/project.cgi/web/or1k/gnu_toolchain_p

Annexes

Ici, vous trouverez dans l'ordre :

- data2asm.pl: page 88.
- data.S: page 89.
- fir.S: page 90.
- reset.S: page 92.
- spr_defs.h: page 95.
- board.h page 102.
- mc.h page 104.
- Makefile page 106.
- fir.ld page 107.
- sim.cfg page 108.
- syms.txt page 109.
- disass.txt page 110.
- install_or1k_uclinux.sh page 113.
- dump2wintxt.pl page 118

data2asm.pl 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
#!/usr/bin/perl
#Class:  section.
#Author: El Mehdi Taïleb.
#Date:   04/15/2004.
#Rev:    0.6b
#Description: Generates an assembly file from a coef and a signal file.
#             Coefs are put inside the .coefs section and the signal inside the .input_sig sectio
n


$coefs_file=$ARGV[0];
$signal_file=$ARGV[1];
$asm_file=$ARGV[2];
#Solving the DOS/UNIX incompatibility in line feed characters!
system "dos2unix $coefs_file";
system "dos2unix $signal_file";
#Getting data
open(Coefs,$coefs_file);
@Coefs=<Coefs>;
close(Coefs);
open(Signal,$signal_file);
@Signal=<Signal>;
close(Signal);
chop(@Coefs);
chop(@Signal);
#Converting data
$Coefs='.hword 0b' . join(',0b',@Coefs);
$Signal='.hword 0b' . join(',0b',@Signal);

#Now, generating the final asm file...
open(ASM,">$asm_file");

print ASM << 'header';
/*This file was automatically generated by some tool from Taïleb El Mehdi!
It contains some Coefs an a signal to be used by some prog of FIR filtering
Just assemble this file and link it with the FIR prog*/
header

print ASM << "asm";
\t.section .coefs ,"a"
\t$Coefs
\t.section .input_sig , "a"
\t$Signal
asm
close(ASM);

print "\nDone...\n";
```

Subset of: data.S 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
/*This file was automatically generated by some tool from Taïleb El Mehdi!
It contains some Coefs an a signal to be used by some prog of FIR filtering
Just assemble this file and link it with the FIR prog*/
        .section .coefs ,"a"
        .hword 0b1111111111000100,0b1111111101001100,0b1111111101011001,0b0000000011101001,0b0000
001101111000,0b0000010001101111,0b0000000111111001,0b1111111001010110,0b1111111000001001,0b000000
0110001111,0b0000001110001110,0b0000000000011010,0b1111101110101110,0b1111110110110111,0b00000100
01110101,0b0000010101010000,0b1111110011011100,0b1111011100111100,0b1111111110111010,0b0000110000
111011,0b0000011100000011,0b1111000011001110,0b1110101011101101,0b0001000100110001,0b010011110101
1110,0b0110111000011010,0b0100111101011110,0b0001000100110001,0b1110101011101101,0b11110000110011
10,0b0000011100000011,0b0000110000111011,0b1111111110111010,0b1111011100111100,0b1111110011011100
,0b0000010101010000,0b0000010001110101,0b1111110110110111,0b1111101110101110,0b0000000000011010,0
b0000001110001110,0b0000000110001111,0b1111111000001001,0b1111111001010110,0b0000000111111001,0b0
000010001101111,0b0000001101111000,0b0000000011101001,0b1111111101011001,0b1111111101001100,0b111
1111111000100
        .section .input_sig , "a"
        .hword 0b1100100010100001,0b1000000000000000,0b0001000000001010,0b0010010011010010,0b1000
000000000000,0b0111111111111111,0b0111111111111111,0b1111101100101110,0b0010100111100100,0b000101
1001011010,0b1110100000011001,0b0101110011100110,0b1011010010110010,0b0111111111111111,0b11101110
10001010,0b0000111010010101,0b0111111111111111,0b0000011110010110,0b1111001111000001,0b1001010101
110101,0b0010010110101111,0b1000000000000000,0b0101101101101110,0b0111111111111111,0b101001110111
0011,0b0110110111010010,0b0111111111111111,0b1000000000000000,0b1000000000000000,0b01001001000110
11,0b1100110011010000,0b0101100001010001,0b0110100001100110,0b0101101100011111,0b0111111111111111
,0b0101010110010100,0b0111111111111111,0b1000000000000000,0b1111110101110111,0b1110101111110000,0
b1000000000000000,0b0010000011101111,0b1000000000000000,0b0111111111111111,0b1001100011110010,0b0
100001110101101,0b0001110000010010,0b1000100111111111,0b1000000000000000,0b1111100001101100,0b100
0000000000000,0b0100111010100110,0b0100000001111110
```

```
#inlude "fir.h"
        /*Important symbols are:
                FIR_BCOEFS      Pointer to the begining of coefs
                FIR_BDATA       Pointer to the begining of data (x)
                FIR_EDATA       Pointer to the end of data (x)
                FIR_BOUT        Pointer to the begining of output (y)
                FIR_LENGTH      Pointer to the length of the filter
        The data pointed by these should be copied to GPRs for efficiency (minimizing access to R
AM)
        Internal registers usage:
                FIR_BCOEFS      R12
                FIR_BDATA       R13 (moves as the filter filters!)
                FIR_BOUT        R14
                FIR_LEGTH       R15
                FIR_EDATA       R24
        Not symbols:
                FIR_PCCOEF      R16
                FIR_PCDATA      R17
                FIR_PCOUT       R18
                FIR_CCOEF       R19
                FIR_CDATA       R20
                FIR_COUT        R21
                FIR_CNTI        R22
                FIR_CMUL        R23
        */
.global _main
        .section .fir
        l.nop
        l.nop
        /*initializing the filter*/
_main:
        l.movhi r15,hi(FIR_LENGTH)
        l.ori   r15,r15,lo(FIR_LENGTH)
        l.movhi r12,hi(FIR_BCOEFS)
        l.ori   r12,r12,lo(FIR_BCOEFS)
        l.movhi r13,hi(FIR_BDATA)
        l.ori   r13,r13,lo(FIR_BDATA)
        l.movhi r14,hi(FIR_BOUT)
        l.ori   r14,r14,lo(FIR_BOUT)
        l.movhi r24,hi(FIR_EDATA)
        l.ori   r24,r24,lo(FIR_EDATA)
        l.addi  r16,r12,0x0
        l.addi  r17,r13,0x0
        l.addi  r18,r14,0x0
        /*Offset pointers*/
        l.addi  r18,r18,-4
        l.addi  r13,r13,-2
        /*Here goes the filtering algo. */
_next_data:
        /*Reseting */
        l.addi  r23,r0,0x0
        l.addi  r22,r0,0x2
        l.addi  r21,r0,0x0
        /*Did we reach the end of data?*/
        l.sfeq  r17,r24
        l.bf    _dump_results /* Yes! we did*/
        l.nop
        /*Point the new begining of data (x), and load it*/
        l.addi  r13,r13,2
        l.addi  r17,r13,0
        l.lhs   r20,0(r17)
        /*Pointing the next y*/
        l.addi  r18,r18,0x4
        /*Return to the begining of the coefs (and load it!)*/
        l.add   r16,r0,r12
        l.lhs   r19,0(r16)

_next_x:
        /*Begin Macking!*/
        /*Multiply*/
        l.mul   r23,r19,r20
```

```
        /*Put here some other actions because of the 3 clock period mul latency*/
        /*Locating next data and coef (xi and bi)*/
        l.addi  r16,r16,0x2
        l.addi  r17,r17,0x2
        /*Loading next data and coef*/
        l.lhs   r19,0(r16)
        l.lhs   r20,0(r17)

        /*Continue macking since multiply result is surely ready*/
        /*Accumulate*/
        l.add   r21,r21,r23

        /*Is this the last mac?*/
        l.sfeq  r22,r15
        l.bf    _next_data
        /*Store the current 'y' to memory. BE CAREFUL, THIS IS A DELAY SLOT, SO IT'S ALWAYS EXECU
TED :-)*/
        l.sw    0(r18),r21

        /*No, this isn't the last mac. So, increment the i and jump to the next sample*/
        l.j     _next_x
        l.addi  r22,r22,0x2

_dump_results:
        l.nop
        l.j     _dump_results
        l.nop
```

reset.S **1/3**
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
#include "spr_defs.h"
#include "board.h"
#include "mc.h"
.global _main
        .section        .stack
        .space  0x10000
_stack:

        .section        .reset,"ax"
        .org 0x100
_reset:
        l.nop
        l.nop
        /*This should be done since the GPRs are not emptyed after reset*/
        l.addi  r2,r0,0x0
        l.addi  r3,r0,0x0
        l.addi  r4,r0,0x0
        l.addi  r5,r0,0x0
        l.addi  r6,r0,0x0
        l.addi  r7,r0,0x0
        l.addi  r8,r0,0x0
        l.addi  r9,r0,0x0
        l.addi  r10,r0,0x0
        l.addi  r11,r0,0x0
        l.addi  r12,r0,0x0
        l.addi  r13,r0,0x0
        l.addi  r14,r0,0x0
        l.addi  r15,r0,0x0
        l.addi  r16,r0,0x0
        l.addi  r17,r0,0x0
        l.addi  r18,r0,0x0
        l.addi  r19,r0,0x0
        l.addi  r20,r0,0x0
        l.addi  r21,r0,0x0
        l.addi  r22,r0,0x0
        l.addi  r23,r0,0x0
        l.addi  r24,r0,0x0
        l.addi  r25,r0,0x0
        l.addi  r26,r0,0x0
        l.addi  r27,r0,0x0
        l.addi  r28,r0,0x0
        l.addi  r29,r0,0x0
        l.addi  r30,r0,0x0
        l.addi  r31,r0,0x0

        l.movhi r3,hi(MC_BASE_ADDR)
        l.ori   r3,r3,MC_BA_MASK
        l.addi  r5,r0,0x0
        l.sw    0(r3),r5
        l.movhi r3,hi(_start)
        l.ori   r3,r3,lo(_start)
        l.jr    r3
        l.nop

        .section        .text
_start:
        l.jal   _init_mc
        l.nop
        l.jal   _init_mem
        l.nop
        /* Set stack pointer */
        l.movhi r1,hi(_stack)
        l.ori   r1,r1,lo(_stack)

        /* Jump to main */
        l.movhi r2,hi(_main)
        l.ori   r2,r2,lo(_main)
        l.jr    r2
        l.nop

_init_mc:
```

```
        l.movhi r3,hi(MC_BASE_ADDR)
        l.ori   r3,r3,lo(MC_BASE_ADDR)

        l.addi  r4,r3,MC_CSC(0)
        l.movhi r5,hi(FLASH_BASE_ADDR)
        l.srai  r5,r5,6
        l.ori   r5,r5,0x0025
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_TMS(0)
        l.movhi r5,hi(FLASH_TMS_VAL)
        l.ori   r5,r5,lo(FLASH_TMS_VAL)
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_BA_MASK
        l.addi  r5,r0,MC_MASK_VAL
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_CSR
        l.movhi r5,hi(MC_CSR_VAL)
        l.ori   r5,r5,lo(MC_CSR_VAL)
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_TMS(1)
        l.movhi r5,hi(SDRAM_TMS_VAL)
        l.ori   r5,r5,lo(SDRAM_TMS_VAL)
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_CSC(1)
        l.movhi r5,hi(SDRAM_BASE_ADDR)
        l.srai  r5,r5,6
        l.ori   r5,r5,0x0411
        l.sw    0(r4),r5

/*          l.addi  r4,r3,MC_TMS(2)
        l.movhi r5,hi(SDRAM_TMS_VAL)
        l.ori   r5,r5,lo(SDRAM_TMS_VAL)
        l.sw    0(r4),r5

        l.addi  r4,r3,MC_CSC(2)
        l.movhi r5,hi(0x00800000)
        l.srai  r5,r5,6
        l.ori   r5,r5,0x0411
        l.sw    0(r4),r5*/

        l.jr    r9
        l.nop


_init_mem:
        /* Wait for SDRAM */
        l.addi  r3,r0,0x2/*0x1000*/
1:      l.sfeqi r3,0
        l.bnf   1b
        l.addi  r3,r3,-1

        /* Copy from flash to sram
        l.movhi r3,hi(_src_beg)
        l.ori   r3,r3,lo(_src_beg)
        l.movhi r4,hi(_icm_start)
        l.ori   r4,r4,lo(_icm_start)
        l.movhi r5,hi(_icm_end)
        l.ori   r5,r5,lo(_icm_end)
        l.sub   r5,r5,r4
        l.sfeqi r5,0
        l.bf    20f
        l.nop
10:     l.lwz   r6,0(r3)
        l.sw    0(r4),r6
        l.addi  r3,r3,4
        l.addi  r4,r4,4
        l.addi  r5,r5,-4
```

```
        l.sfgtsi r5,0
        l.bf    10b
        l.nop
20:*/

        /* Copy from flash to sram */
        l.movhi r3,hi(_src_beg)
        l.ori   r3,r3,lo(_src_beg)
        l.movhi r4,hi(_vec_start)
        l.ori   r4,r4,lo(_vec_start)
        l.movhi r5,hi(_vec_end)
        l.ori   r5,r5,lo(_vec_end)
        l.sub   r5,r5,r4
        l.sfeqi r5,0
        l.bf    2f
        l.nop
1:      l.lwz   r6,0(r3)
        l.sw    0(r4),r6
        l.addi  r3,r3,4
        l.addi  r4,r4,4
        l.addi  r5,r5,-4
        l.sfgtsi r5,0
        l.bf    1b
        l.nop
2:
        l.movhi r4,hi(_dst_beg)
        l.ori   r4,r4,lo(_dst_beg)
        l.movhi r5,hi(_dst_end)
        l.ori   r5,r5,lo(_dst_end)
1:      l.sfgeu r4,r5
        l.bf    1f
        l.nop
        l.lwz   r8,0(r3)
        l.sw    0(r4),r8
        l.addi  r3,r3,4
        l.bnf   1b
        l.addi  r4,r4,4
1:
        l.addi  r3,r0,0
        l.addi  r4,r0,0
        l.jr    r9
```

```
/* spr_defs.h -- Defines OR1K architecture specific special-purpose registers
   Copyright (C) 1999 Damjan Lampret, lampret@opencores.org

This file is part of OpenRISC 1000 Architectural Simulator.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA. */

/* This file is also used by microkernel test bench. Among
others it is also used in assembly file(s). */

/* Definition of special-purpose registers (SPRs) */

#define MAX_GRPS (32)
#define MAX_SPRS_PER_GRP_BITS (11)
#define MAX_SPRS_PER_GRP (1 << MAX_SPRS_PER_GRP_BITS)
#define MAX_SPRS (0x10000)

/* Base addresses for the groups */
#define SPRGROUP_SYS    (0<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_DMMU   (1<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_IMMU   (2<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_DC     (3<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_IC     (4<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_MAC    (5<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_D      (6<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_PC     (7<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_PM     (8<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_PIC    (9<< MAX_SPRS_PER_GRP_BITS)
#define SPRGROUP_TT     (10<< MAX_SPRS_PER_GRP_BITS)

/* System control and status group */
#define SPR_VR          (SPRGROUP_SYS + 0)
#define SPR_UPR         (SPRGROUP_SYS + 1)
#define SPR_CPUCFGR     (SPRGROUP_SYS + 2)
#define SPR_DMMUCFGR    (SPRGROUP_SYS + 3)
#define SPR_IMMUCFGR    (SPRGROUP_SYS + 4)
#define SPR_DCCFGR      (SPRGROUP_SYS + 5)
#define SPR_ICCFGR      (SPRGROUP_SYS + 6)
#define SPR_DCFGR       (SPRGROUP_SYS + 7)
#define SPR_PCCFGR      (SPRGROUP_SYS + 8)
#define SPR_NPC         (SPRGROUP_SYS + 16)  /* CZ 21/06/01 */
#define SPR_SR          (SPRGROUP_SYS + 17)  /* CZ 21/06/01 */
#define SPR_PPC         (SPRGROUP_SYS + 18)  /* CZ 21/06/01 */
#define SPR_EPCR_BASE   (SPRGROUP_SYS + 32)  /* CZ 21/06/01 */
#define SPR_EPCR_LAST   (SPRGROUP_SYS + 47)  /* CZ 21/06/01 */
#define SPR_EEAR_BASE   (SPRGROUP_SYS + 48)
#define SPR_EEAR_LAST   (SPRGROUP_SYS + 63)
#define SPR_ESR_BASE    (SPRGROUP_SYS + 64)
#define SPR_ESR_LAST    (SPRGROUP_SYS + 79)

#if 0
/* Data MMU group */
#define SPR_DMMUCR      (SPRGROUP_DMMU + 0)
#define SPR_DTLBMR_BASE(WAY)    (SPRGROUP_DMMU + 0x200 + (WAY) * 0x200)
#define SPR_DTLBMR_LAST(WAY)    (SPRGROUP_DMMU + 0x2ff + (WAY) * 0x200)
#define SPR_DTLBTR_BASE(WAY)    (SPRGROUP_DMMU + 0x300 + (WAY) * 0x200)
#define SPR_DTLBTR_LAST(WAY)    (SPRGROUP_DMMU + 0x3ff + (WAY) * 0x200)

/* Instruction MMU group */
#define SPR_IMMUCR      (SPRGROUP_IMMU + 0)
```

```
#define SPR_ITLBMR_BASE(WAY)      (SPRGROUP_IMMU + 0x200 + (WAY) * 0x200)
#define SPR_ITLBMR_LAST(WAY)      (SPRGROUP_IMMU + 0x2ff + (WAY) * 0x200)
#define SPR_ITLBTR_BASE(WAY)      (SPRGROUP_IMMU + 0x300 + (WAY) * 0x200)
#define SPR_ITLBTR_LAST(WAY)      (SPRGROUP_IMMU + 0x3ff + (WAY) * 0x200)
#else
/* Data MMU group */
#define SPR_DMMUCR  (SPRGROUP_DMMU + 0)
#define SPR_DTLBMR_BASE(WAY)    (SPRGROUP_DMMU + 0x200 + (WAY) * 0x100)
#define SPR_DTLBMR_LAST(WAY)    (SPRGROUP_DMMU + 0x27f + (WAY) * 0x100)
#define SPR_DTLBTR_BASE(WAY)    (SPRGROUP_DMMU + 0x280 + (WAY) * 0x100)
#define SPR_DTLBTR_LAST(WAY)    (SPRGROUP_DMMU + 0x2ff + (WAY) * 0x100)

/* Instruction MMU group */
#define SPR_IMMUCR  (SPRGROUP_IMMU + 0)
#define SPR_ITLBMR_BASE(WAY)    (SPRGROUP_IMMU + 0x200 + (WAY) * 0x100)
#define SPR_ITLBMR_LAST(WAY)    (SPRGROUP_IMMU + 0x27f + (WAY) * 0x100)
#define SPR_ITLBTR_BASE(WAY)    (SPRGROUP_IMMU + 0x280 + (WAY) * 0x100)
#define SPR_ITLBTR_LAST(WAY)    (SPRGROUP_IMMU + 0x2ff + (WAY) * 0x100)
#endif
/* Data cache group */
#define SPR_DCCR            (SPRGROUP_DC + 0)
#define SPR_DCBPR           (SPRGROUP_DC + 1)
#define SPR_DCBFR           (SPRGROUP_DC + 2)
#define SPR_DCBIR           (SPRGROUP_DC + 3)
#define SPR_DCBWR           (SPRGROUP_DC + 4)
#define SPR_DCBLR           (SPRGROUP_DC + 5)
#define SPR_DCR_BASE(WAY)          (SPRGROUP_DC + 0x200 + (WAY) * 0x200)
#define SPR_DCR_LAST(WAY)          (SPRGROUP_DC + 0x3ff + (WAY) * 0x200)

/* Instruction cache group */
#define SPR_ICCR            (SPRGROUP_IC + 0)
#define SPR_ICBPR           (SPRGROUP_IC + 1)
#define SPR_ICBIR           (SPRGROUP_IC + 2)
#define SPR_ICBLR           (SPRGROUP_IC + 3)
#define SPR_ICR_BASE(WAY)          (SPRGROUP_IC + 0x200 + (WAY) * 0x200)
#define SPR_ICR_LAST(WAY)          (SPRGROUP_IC + 0x3ff + (WAY) * 0x200)

/* MAC group */
#define SPR_MACLO           (SPRGROUP_MAC + 1)
#define SPR_MACHI           (SPRGROUP_MAC + 2)

/* Debug group */
#define SPR_DVR(N)          (SPRGROUP_D + (N))
#define SPR_DCR(N)          (SPRGROUP_D + 8 + (N))
#define SPR_DMR1            (SPRGROUP_D + 16)
#define SPR_DMR2            (SPRGROUP_D + 17)
#define SPR_DWCR0           (SPRGROUP_D + 18)
#define SPR_DWCR1           (SPRGROUP_D + 19)
#define SPR_DSR             (SPRGROUP_D + 20)
#define SPR_DRR             (SPRGROUP_D + 21)

/* Performance counters group */
#define SPR_PCCR(N)         (SPRGROUP_PC + (N))
#define SPR_PCMR(N)         (SPRGROUP_PC + 8 + (N))

/* Power management group */
#define SPR_PMR (SPRGROUP_PM + 0)

/* PIC group */
#define SPR_PICMR (SPRGROUP_PIC + 0)
#define SPR_PICPR (SPRGROUP_PIC + 1)
#define SPR_PICSR (SPRGROUP_PIC + 2)

/* Tick Timer group */
#define SPR_TTMR (SPRGROUP_TT + 0)
#define SPR_TTCR (SPRGROUP_TT + 1)

/*
 * Bit definitions for the Version Register
 *
 */
#define SPR_VR_VER          0xffff0000  /* Processor version */
```

```
#define SPR_VR_REV       0x0000003f  /* Processor revision */

/*
 * Bit definitions for the Unit Present Register
 *
 */
#define SPR_UPR_UP       0x00000001  /* UPR present */
#define SPR_UPR_DCP      0x00000002  /* Data cache present */
#define SPR_UPR_ICP      0x00000004  /* Instruction cache present */
#define SPR_UPR_DMP      0x00000008  /* Data MMU present */
#define SPR_UPR_IMP      0x00000010  /* Instruction MMU present */
#define SPR_UPR_OB32P    0x00000020  /* ORBIS32 present */
#define SPR_UPR_OB64P    0x00000040  /* ORBIS64 present */
#define SPR_UPR_OF32P    0x00000080  /* ORFPX32 present */
#define SPR_UPR_OF64P    0x00000100  /* ORFPX64 present */
#define SPR_UPR_OV32P    0x00000200  /* ORVDX32 present */
#define SPR_UPR_OV64P    0x00000400  /* ORVDX64 present */
#define SPR_UPR_DUP      0x00000800  /* Debug unit present */
#define SPR_UPR_PCUP     0x00001000  /* Performance counters unit present */
#define SPR_UPR_PMP      0x00002000  /* Power management present */
#define SPR_UPR_PICP     0x00004000  /* PIC present */
#define SPR_UPR_TTP      0x00008000  /* Tick timer present */
#define SPR_UPR_SRP      0x00010000  /* Shadow registers present */
#define SPR_UPR_RES      0x00fe0000  /* ORVDX32 present */
#define SPR_UPR_CUST     0xff000000  /* Custom units */

/*
 * Bit definitions for the Supervision Register
 *
 */
#define SPR_SR_CID       0xf0000000  /* Context ID */
#define SPR_SR_FO        0x00008000  /* Fixed one */
#define SPR_SR_EPH       0x00004000  /* Exception Prefixi High */
#define SPR_SR_DSX       0x00002000  /* Delay Slot Exception */
#define SPR_SR_OVE       0x00001000  /* Overflow flag Exception */
#define SPR_SR_OV        0x00000800  /* Overflow flag */
#define SPR_SR_CY        0x00000400  /* Carry flag */
#define SPR_SR_F         0x00000200  /* Condition Flag */
#define SPR_SR_CE        0x00000100  /* CID Enable */
#define SPR_SR_LEE       0x00000080  /* Little Endian Enable */
#define SPR_SR_IME       0x00000040  /* Instruction MMU Enable */
#define SPR_SR_DME       0x00000020  /* Data MMU Enable */
#define SPR_SR_ICE       0x00000010  /* Instruction Cache Enable */
#define SPR_SR_DCE       0x00000008  /* Data Cache Enable */
#define SPR_SR_IEE       0x00000004  /* Interrupt Exception Enable */
#define SPR_SR_TEE       0x00000002  /* Tick timer Exception Enable */
#define SPR_SR_SM        0x00000001  /* Supervisor Mode */

/*
 * Bit definitions for the Data MMU Control Register
 *
 */
#define SPR_DMMUCR_P2S   0x0000003e  /* Level 2 Page Size */
#define SPR_DMMUCR_P1S   0x000007c0  /* Level 1 Page Size */
#define SPR_DMMUCR_VADDR_WIDTH  0x0000f800  /* Virtual ADDR Width */
#define SPR_DMMUCR_PADDR_WIDTH  0x000f0000  /* Physical ADDR Width */

/*
 * Bit definitions for the Instruction MMU Control Register
 *
 */
#define SPR_IMMUCR_P2S   0x0000003e  /* Level 2 Page Size */
#define SPR_IMMUCR_P1S   0x000007c0  /* Level 1 Page Size */
#define SPR_IMMUCR_VADDR_WIDTH  0x0000f800  /* Virtual ADDR Width */
#define SPR_IMMUCR_PADDR_WIDTH  0x000f0000  /* Physical ADDR Width */

/*
 * Bit definitions for the Data TLB Match Register
 *
 */
#define SPR_DTLBMR_V     0x00000001  /* Valid */
#define SPR_DTLBMR_PL1   0x00000002  /* Page Level 1 (if 0 then PL2) */
```

```
#define SPR_DTLBMR_CID  0x0000003c  /* Context ID */
#define SPR_DTLBMR_LRU  0x000000c0  /* Least Recently Used */
#define SPR_DTLBMR_VPN  0xfffff000  /* Virtual Page Number */

/*
 * Bit definitions for the Data TLB Translate Register
 *
 */
#define SPR_DTLBTR_CC   0x00000001  /* Cache Coherency */
#define SPR_DTLBTR_CI   0x00000002  /* Cache Inhibit */
#define SPR_DTLBTR_WBC  0x00000004  /* Write-Back Cache */
#define SPR_DTLBTR_WOM  0x00000008  /* Weakly-Ordered Memory */
#define SPR_DTLBTR_A    0x00000010  /* Accessed */
#define SPR_DTLBTR_D    0x00000020  /* Dirty */
#define SPR_DTLBTR_URE  0x00000040  /* User Read Enable */
#define SPR_DTLBTR_UWE  0x00000080  /* User Write Enable */
#define SPR_DTLBTR_SRE  0x00000100  /* Supervisor Read Enable */
#define SPR_DTLBTR_SWE  0x00000200  /* Supervisor Write Enable */
#define SPR_DTLBTR_PPN  0xfffff000  /* Physical Page Number */
#define DTLB_PR_NOLIMIT  (SPR_DTLBTR_URE  | \
                          SPR_DTLBTR_UWE  | \
                          SPR_DTLBTR_SRE  | \
                          SPR_DTLBTR_SWE  )
/*
 * Bit definitions for the Instruction TLB Match Register
 *
 */
#define SPR_ITLBMR_V    0x00000001  /* Valid */
#define SPR_ITLBMR_PL1  0x00000002  /* Page Level 1 (if 0 then PL2) */
#define SPR_ITLBMR_CID  0x0000003c  /* Context ID */
#define SPR_ITLBMR_LRU  0x000000c0  /* Least Recently Used */
#define SPR_ITLBMR_VPN  0xfffff000  /* Virtual Page Number */

/*
 * Bit definitions for the Instruction TLB Translate Register
 *
 */
#define SPR_ITLBTR_CC   0x00000001  /* Cache Coherency */
#define SPR_ITLBTR_CI   0x00000002  /* Cache Inhibit */
#define SPR_ITLBTR_WBC  0x00000004  /* Write-Back Cache */
#define SPR_ITLBTR_WOM  0x00000008  /* Weakly-Ordered Memory */
#define SPR_ITLBTR_A    0x00000010  /* Accessed */
#define SPR_ITLBTR_D    0x00000020  /* Dirty */
#define SPR_ITLBTR_SXE  0x00000040  /* User Read Enable */
#define SPR_ITLBTR_UXE  0x00000080  /* User Write Enable */
#define SPR_ITLBTR_PPN  0xfffff000  /* Physical Page Number */
#define ITLB_PR_NOLIMIT  (SPR_ITLBTR_SXE  | \
                          SPR_ITLBTR_UXE  )


/*
 * Bit definitions for Data Cache Control register
 *
 */
#define SPR_DCCR_EW     0x000000ff  /* Enable ways */

/*
 * Bit definitions for Insn Cache Control register
 *
 */
#define SPR_ICCR_EW     0x000000ff  /* Enable ways */

/*
 * Bit definitions for Debug Control registers
 *
 */
#define SPR_DCR_DP      0x00000001  /* DVR/DCR present */
#define SPR_DCR_CC      0x0000000e  /* Compare condition */
#define SPR_DCR_SC      0x00000010  /* Signed compare */
#define SPR_DCR_CT      0x000000e0  /* Compare to */

/* Bit results with SPR_DCR_CC mask */
```

```
#define SPR_DCR_CC_MASKED 0x00000000
#define SPR_DCR_CC_EQUAL  0x00000001
#define SPR_DCR_CC_LESS   0x00000002
#define SPR_DCR_CC_LESSE  0x00000003
#define SPR_DCR_CC_GREAT  0x00000004
#define SPR_DCR_CC_GREATE 0x00000005
#define SPR_DCR_CC_NEQUAL 0x00000006

/* Bit results with SPR_DCR_CT mask */
#define SPR_DCR_CT_DISABLED 0x00000000
#define SPR_DCR_CT_IFEA     0x00000020
#define SPR_DCR_CT_LEA      0x00000040
#define SPR_DCR_CT_SEA      0x00000060
#define SPR_DCR_CT_LD       0x00000080
#define SPR_DCR_CT_SD       0x000000a0
#define SPR_DCR_CT_LSEA     0x000000c0

/*
 * Bit definitions for Debug Mode 1 register
 *
 */
#define SPR_DMR1_CW0    0x00000003  /* Chain watchpoint 0 */
#define SPR_DMR1_CW1    0x0000000c  /* Chain watchpoint 1 */
#define SPR_DMR1_CW2    0x00000030  /* Chain watchpoint 2 */
#define SPR_DMR1_CW3    0x000000c0  /* Chain watchpoint 3 */
#define SPR_DMR1_CW4    0x00000300  /* Chain watchpoint 4 */
#define SPR_DMR1_CW5    0x00000c00  /* Chain watchpoint 5 */
#define SPR_DMR1_CW6    0x00003000  /* Chain watchpoint 6 */
#define SPR_DMR1_CW7    0x0000c000  /* Chain watchpoint 7 */
#define SPR_DMR1_CW8    0x00030000  /* Chain watchpoint 8 */
#define SPR_DMR1_CW9    0x000c0000  /* Chain watchpoint 9 */
#define SPR_DMR1_CW10   0x00300000  /* Chain watchpoint 10 */
#define SPR_DMR1_ST     0x00400000  /* Single-step trace*/
#define SPR_DMR1_BT     0x00800000  /* Branch trace */
#define SPR_DMR1_DXFW   0x01000000  /* Disable external force watchpoint */

/*
 * Bit definitions for Debug Mode 2 register
 *
 */
#define SPR_DMR2_WCE0   0x00000001  /* Watchpoint counter 0 enable */
#define SPR_DMR2_WCE1   0x00000002  /* Watchpoint counter 0 enable */
#define SPR_DMR2_AWTC   0x00001ffc  /* Assign watchpoints to counters */
#define SPR_DMR2_WGB    0x00ffe000  /* Watchpoints generating breakpoint */

/*
 * Bit definitions for Debug watchpoint counter registers
 *
 */
#define SPR_DWCR_COUNT  0x0000ffff  /* Count */
#define SPR_DWCR_MATCH  0xffff0000  /* Match */

/*
 * Bit definitions for Debug stop register
 *
 */
#define SPR_DSR_RSTE    0x00000001  /* Reset exception */
#define SPR_DSR_BUSEE   0x00000002  /* Bus error exception */
#define SPR_DSR_DPFE    0x00000004  /* Data Page Fault exception */
#define SPR_DSR_IPFE    0x00000008  /* Insn Page Fault exception */
#define SPR_DSR_TTE     0x00000010  /* iTick Timer exception */
#define SPR_DSR_AE      0x00000020  /* Alignment exception */
#define SPR_DSR_IIE     0x00000040  /* Illegal Instruction exception */
#define SPR_DSR_IE      0x00000080  /* Interrupt exception */
#define SPR_DSR_DME     0x00000100  /* DTLB miss exception */
#define SPR_DSR_IME     0x00000200  /* ITLB miss exception */
#define SPR_DSR_RE      0x00000400  /* Range exception */
#define SPR_DSR_SCE     0x00000800  /* System call exception */
#define SPR_DSR_SSE     0x00001000  /* Single Step Exception */
#define SPR_DSR_TE      0x00002000  /* Trap exception */

/*
```

```
 * Bit definitions for Debug reason register
 *
 */
#define SPR_DRR_RSTE    0x00000001  /* Reset exception */
#define SPR_DRR_BUSEE   0x00000002  /* Bus error exception */
#define SPR_DRR_DPFE    0x00000004  /* Data Page Fault exception */
#define SPR_DRR_IPFE    0x00000008  /* Insn Page Fault exception */
#define SPR_DRR_TTE     0x00000010  /* Tick Timer exception */
#define SPR_DRR_AE      0x00000020  /* Alignment exception */
#define SPR_DRR_IIE     0x00000040  /* Illegal Instruction exception */
#define SPR_DRR_IE      0x00000080  /* Interrupt exception */
#define SPR_DRR_DME     0x00000100  /* DTLB miss exception */
#define SPR_DRR_IME     0x00000200  /* ITLB miss exception */
#define SPR_DRR_RE      0x00000400  /* Range exception */
#define SPR_DRR_SCE     0x00000800  /* System call exception */
#define SPR_DRR_TE      0x00001000  /* Trap exception */

/*
 * Bit definitions for Performance counters mode registers
 *
 */
#define SPR_PCMR_CP     0x00000001  /* Counter present */
#define SPR_PCMR_UMRA   0x00000002  /* User mode read access */
#define SPR_PCMR_CISM   0x00000004  /* Count in supervisor mode */
#define SPR_PCMR_CIUM   0x00000008  /* Count in user mode */
#define SPR_PCMR_LA     0x00000010  /* Load access event */
#define SPR_PCMR_SA     0x00000020  /* Store access event */
#define SPR_PCMR_IF     0x00000040  /* Instruction fetch event*/
#define SPR_PCMR_DCM    0x00000080  /* Data cache miss event */
#define SPR_PCMR_ICM    0x00000100  /* Insn cache miss event */
#define SPR_PCMR_IFS    0x00000200  /* Insn fetch stall event */
#define SPR_PCMR_LSUS   0x00000400  /* LSU stall event */
#define SPR_PCMR_BS     0x00000800  /* Branch stall event */
#define SPR_PCMR_DTLBM  0x00001000  /* DTLB miss event */
#define SPR_PCMR_ITLBM  0x00002000  /* ITLB miss event */
#define SPR_PCMR_DDS    0x00004000  /* Data dependency stall event */
#define SPR_PCMR_WPE    0x03ff8000  /* Watchpoint events */

/*
 * Bit definitions for the Power management register
 *
 */
#define SPR_PMR_SDF     0x0000000f  /* Slow down factor */
#define SPR_PMR_DME     0x00000010  /* Doze mode enable */
#define SPR_PMR_SME     0x00000020  /* Sleep mode enable */
#define SPR_PMR_DCGE    0x00000040  /* Dynamic clock gating enable */
#define SPR_PMR_SUME    0x00000080  /* Suspend mode enable */

/*
 * Bit definitions for PICMR
 *
 */
#define SPR_PICMR_IUM   0xfffffffc  /* Interrupt unmask */

/*
 * Bit definitions for PICPR
 *
 */
#define SPR_PICPR_IPRIO 0xfffffffc  /* Interrupt priority */

/*
 * Bit definitions for PICSR
 *
 */
#define SPR_PICSR_IS    0xffffffff  /* Interrupt status */

/*
 * Bit definitions for Tick Timer Control Register
 *
 */
#define SPR_TTCR_PERIOD 0x0fffffff  /* Time Period */
#define SPR_TTMR_PERIOD SPR_TTCR_PERIOD
```

```
#define SPR_TTMR_IP      0x10000000  /* Interrupt Pending */
#define SPR_TTMR_IE      0x20000000  /* Interrupt Enable */
#define SPR_TTMR_RT      0x40000000  /* Restart tick */
#define SPR_TTMR_SR      0x80000000  /* Single run */
#define SPR_TTMR_CR      0xc0000000  /* Continuous run */
#define SPR_TTMR_M       0xc0000000  /* Tick mode */

/*
 * l.nop constants
 *
 */
#define NOP_NOP          0x0000      /* Normal nop instruction */
#define NOP_EXIT         0x0001      /* End of simulation */
#define NOP_REPORT       0x0002      /* Simple report */
#define NOP_PRINTF       0x0003      /* Simprintf instruction */
#define NOP_REPORT_FIRST 0x0400      /* Report with number */
#define NOP_REPORT_LAST  0x03ff      /* Report with number */
```

board.h 1/2
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
#ifndef _BOARD_H_
#define _BOARD_H_

#ifdef XESS
#define MC_ENABLED          0
#else
#define MC_ENABLED          1
#endif

#define IC_ENABLE           0
#define IC_SIZE          8192
#define DC_ENABLE           0
#define DC_SIZE          8192

#define MC_CSR_VAL       0x0B000300
#define MC_MASK_VAL      0x000003ff
#define FLASH_BASE_ADDR  0xf0000000
#define FLASH_SIZE       0x08000000
#define FLASH_BLOCK_SIZE 0x20000
#define FLASH_TMS_VAL    0x19220057
#define SDRAM_BASE_ADDR  0x00000000
#define SDRAM_TMS_VAL    0x00000103

#ifdef XESS
#define IN_CLK              10000000
#else
#define IN_CLK              25000000
#endif

#define TICKS_PER_SEC   100

#define STACK_SIZE          0x10000

#ifdef XESS
#define UART_BAUD_RATE  19200
#else
#define UART_BAUD_RATE  57600
#endif

#define UART_BASE           0x90000000
#define UART_IRQ         19
#ifdef XESS
#define ETH_BASE         0x92000000
#else
#define ETH_BASE         0xD0000000
#endif
#define ETH_IRQ          15
#define MC_BASE_ADDR     0x93000000
#define SPI_BASE         0xa0000000

#ifdef XESS
 #define ETH_DATA_BASE  0x00100000 /*  Address for ETH_DATA */
#else
 #define ETH_DATA_BASE  0xa8000000 /*  Address for ETH_DATA */
#endif

#define BOARD_DEF_IP      0x0a010185
#define BOARD_DEF_MASK   0xff000000
#define BOARD_DEF_GW     0x0a010101

#define ETH_MACADDR0      0x00
#define ETH_MACADDR1      0x12
#define ETH_MACADDR2     0x34
#define ETH_MACADDR3      0x56
#define ETH_MACADDR4     0x78
#define ETH_MACADDR5      0x9a

#define CRT_ENABLED         1
#define CRT_BASE_ADDR    0xc0000000
#define FB_BASE_ADDR      0xa8000000

/* Whether online help is available -- saves space */
```

```
#define HELP_ENABLED     1

/* Whether self check is enabled */
#define SELF_CHECK      1

/* Whether we have keyboard suppport */
#define KBD_ENABLED     1

/* Keyboard base address */
#define KBD_BASE_ADD    0x98000000

#define KBD_IRQ         12

/* Keyboard buffer size */
#define KBDBUF_SIZE     256

/* Which console is used (CT_NONE, CT_SIM, CT_UART, CT_CRT) */
#define CONSOLE_TYPE    CT_UART

#endif
```

```
/* mc.h -- Simulation of Memory Controller
        Copyright (C) 2001 by Marko Mlinar, markom@opencores.org

        This file is part of OpenRISC 1000 Architectural Simulator.

        This program is free software; you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
        the Free Software Foundation; either version 2 of the License, or
        (at your option) any later version.

        This program is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with this program; if not, write to the Free Software
        Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

/* Prototypes */
#ifndef __MC_H
#define __MC_H

#define N_CE         (8)

#define MC_CSR       (0x00)
#define MC_POC       (0x04)
#define MC_BA_MASK   (0x08)
#define MC_CSC(i)    (0x10 + (i) * 8)
#define MC_TMS(i)    (0x14 + (i) * 8)

#define MC_ADDR_SPACE (MC_CSC(N_CE))

/* POC register field definition */
#define MC_POC_EN_BW_OFFSET      0
#define MC_POC_EN_BW_WIDTH       2
#define MC_POC_EN_MEMTYPE_OFFSET         2
#define MC_POC_EN_MEMTYPE_WIDTH 2

/* CSC register field definition */
#define MC_CSC_EN_OFFSET         0
#define MC_CSC_MEMTYPE_OFFSET    1
#define MC_CSC_MEMTYPE_WIDTH     2
#define MC_CSC_BW_OFFSET         4
#define MC_CSC_BW_WIDTH          2
#define MC_CSC_MS_OFFSET         6
#define MC_CSC_MS_WIDTH          2
#define MC_CSC_WP_OFFSET         8
#define MC_CSC_BAS_OFFSET        9
#define MC_CSC_KRO_OFFSET        10
#define MC_CSC_PEN_OFFSET        11
#define MC_CSC_SEL_OFFSET        16
#define MC_CSC_SEL_WIDTH         8

#define MC_CSC_MEMTYPE_SDRAM  0
#define MC_CSC_MEMTYPE_SSRAM  1
#define MC_CSC_MEMTYPE_ASYNC  2
#define MC_CSC_MEMTYPE_SYNC   3

#define MC_CSR_VALID             0xFF000703LU
#define MC_POC_VALID             0x0000000FLU
#define MC_BA_MASK_VALID         0x000000FFLU
#define MC_CSC_VALID             0x00FF0FFFLU
#define MC_TMS_SDRAM_VALID       0x0FFF83FFLU
#define MC_TMS_SSRAM_VALID       0x00000000LU
#define MC_TMS_ASYNC_VALID       0x03FFFFFFLU
#define MC_TMS_SYNC_VALID        0x01FFFFFFLU
#define MC_TMS_VALID             0xFFFFFFFFLU /* reg test compat. */

/* TMS register field definition SDRAM */
#define MC_TMS_SDRAM_TRFC_OFFSET         24
```

mc.h **2/2**
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
#define MC_TMS_SDRAM_TRFC_WIDTH         4
#define MC_TMS_SDRAM_TRP_OFFSET         20
#define MC_TMS_SDRAM_TRP_WIDTH          4
#define MC_TMS_SDRAM_TRCD_OFFSET        17
#define MC_TMS_SDRAM_TRCD_WIDTH         4
#define MC_TMS_SDRAM_TWR_OFFSET         15
#define MC_TMS_SDRAM_TWR_WIDTH          2
#define MC_TMS_SDRAM_WBL_OFFSET         9
#define MC_TMS_SDRAM_OM_OFFSET          7
#define MC_TMS_SDRAM_OM_WIDTH           2
#define MC_TMS_SDRAM_CL_OFFSET          4
#define MC_TMS_SDRAM_CL_WIDTH           3
#define MC_TMS_SDRAM_BT_OFFSET          3
#define MC_TMS_SDRAM_BL_OFFSET          0
#define MC_TMS_SDRAM_BL_WIDTH           3

/* TMS register field definition ASYNC */
#define MC_TMS_ASYNC_TWWD_OFFSET        20
#define MC_TMS_ASYNC_TWWD_WIDTH         6
#define MC_TMS_ASYNC_TWD_OFFSET         16
#define MC_TMS_ASYNC_TWD_WIDTH          4
#define MC_TMS_ASYNC_TWPW_OFFSET        12
#define MC_TMS_ASYNC_TWPW_WIDTH         4
#define MC_TMS_ASYNC_TRDZ_OFFSET        8
#define MC_TMS_ASYNC_TRDZ_WIDTH         4
#define MC_TMS_ASYNC_TRDV_OFFSET        0
#define MC_TMS_ASYNC_TRDV_WIDTH         8

/* TMS register field definition SYNC  */
#define MC_TMS_SYNC_TTO_OFFSET          16
#define MC_TMS_SYNC_TTO_WIDTH           9
#define MC_TMS_SYNC_TWR_OFFSET          12
#define MC_TMS_SYNC_TWR_WIDTH           4
#define MC_TMS_SYNC_TRDZ_OFFSET         8
#define MC_TMS_SYNC_TRDZ_WIDTH          4
#define MC_TMS_SYNC_TRDV_OFFSET         0
#define MC_TMS_SYNC_TRDV_WIDTH          8

#endif
```

Makefile 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
PRJ=my_filter

all: fir.o data.o reset.o
	or32-uclinux-ld -Tfir.ld $? -o $(PRJ).or32
	or32-uclinux-objdump -s $(PRJ).or32 >hexa.txt
	or32-uclinux-objdump -d $(PRJ).or32 >disass.txt
	or32-uclinux-objdump -t $(PRJ).or32 >syms.txt

fir.o: fir.S fir.h
	or32-uclinux-gcc -O2 -c fir.S -o $@

data.o: data.S
	or32-uclinux-as $? -o $@

reset.o: reset.S board.h spr_defs.h mc.h
	or32-uclinux-gcc -O2 -c reset.S -o $@

data.S: $(PRJ).coe $(PRJ).sig
	perl data2asm.pl $(PRJ).coe $(PRJ).sig $@

clean:
	rm -f *~ *.o *.or32
	rm -f data.S
```

fir.ld 1/1

~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
MEMORY
        {
        vectors : ORIGIN = 0x00000000, LENGTH = 0x00002000
        ram     : ORIGIN = 0x00002000, LENGTH = 0x007fe000
        flash   : ORIGIN = 0xf0000000, LENGTH = 0x00800000
        }
SECTIONS
{
        .reset :
        {
        *(.reset)
        }>flash

        .text ALIGN(0x04):
        {
        *(.text)
        *(.fir)
        }>flash

        .coefs ALIGN(0x04):
        {
        FIR_BCOEFS = .;
        *(.coefs)
        }>flash

        .input_sig ALIGN(0x04):
        {
        FIR_BDATA = .;
        *(.input_sig)
        FIR_EDATA = .;
        FIR_LENGTH = SIZEOF (.coefs) ;
        }>flash

      .dummy ALIGN(0x04):
        {
        _src_beg = .;
        } > flash

      .vectors :
        AT ( ADDR (.dummy) )
        {
        _vec_start = .;
        *(.vectors)
        _vec_end = .;
        } > vectors

     .data :
        AT ( ADDR (.dummy) + SIZEOF (.vectors))
        {
        _dst_beg = .;
        *(.data)
        _dst_end = .;
        FIR_BOUT = .;
        } > ram

      .bss (FIR_BOUT + (FIR_EDATA - FIR_BDATA)) :
        {
        *(.bss)
        } > ram

      .stack (NOLOAD) :
        {
        *(.stack)
        _src_addr = .;
        } > ram

}
```

syms.txt 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
my_filter.or32:     file format elf32-or32

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 fir.S
00000000 l    df *ABS*  00000000 <command line>
00000000 l    df *ABS*  00000000 <built-in>
00000000 l    df *ABS*  00000000 fir.S
f00001a0 l    d  .text  00000000
00002000 l    d  .data  00000000
000032c2 l    d  .bss   00000000
f00002d0 l    d  .text  00000000
f0000314 l       .text  00000000 _next_data
f0000370 l       .text  00000000 _dump_results
f0000344 l       .text  00000000 _next_x
f00001a0 l    d  .text  00000000
00002000 l    d  .data  00000000
000032c2 l    d  .bss   00000000
f000037c l    d  .coefs 00000000
f00003e4 l    d  .input_sig     00000000
00000000 l    df *ABS*  00000000 reset.S
00000000 l    df *ABS*  00000000 mc.h
00000000 l    df *ABS*  00000000 reset.S
00000000 l    df *ABS*  00000000 board.h
00000000 l    df *ABS*  00000000 reset.S
00000000 l    df *ABS*  00000000 spr_defs.h
00000000 l    df *ABS*  00000000 reset.S
00000000 l    df *ABS*  00000000 <command line>
00000000 l    df *ABS*  00000000 <built-in>
00000000 l    df *ABS*  00000000 reset.S
f00001a0 l    d  .text  00000000
00002000 l    d  .data  00000000
000032c2 l    d  .bss   00000000
000032c2 l    d  .stack 00000000
000132c2 l       .stack 00000000 _stack
f0000000 l    d  .reset 00000000
f0000100 l       .reset 00000000 _reset
f00001a0 l       .text  00000000 _start
f00001c8 l       .text  00000000 _init_mc
f000023c l       .text  00000000 _init_mem
f00016a8 l    d  .dummy 00000000
00000000 l    d  .vectors       00000000
00000066 g       *ABS*  00000000 FIR_LENGTH
00000000 g       .vectors       00000000 _vec_start
000132c2 g       .stack 00000000 _src_addr
f00016a6 g       .input_sig     00000000 FIR_EDATA
00002000 g       .data  00000000 _dst_end
00002000 g       .data  00000000 FIR_BOUT
f00003e4 g       .input_sig     00000000 FIR_BDATA
00000000 g       .vectors       00000000 _vec_end
00002000 g       .data  00000000 _dst_beg
f000037c g       .coefs 00000000 FIR_BCOEFS
f00016a8 g       .dummy 00000000 _src_beg
f00002d8 g       .text  00000000 _main
```

disass.txt **1/3**
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
my_filter.or32:     file format elf32-or32

Disassembly of section .reset:

f0000000 <_reset-0x100>:
        ...

f0000100 <_reset>:
f0000100:       15 00 00 00     l.nop 0x0
f0000104:       15 00 00 00     l.nop 0x0
f0000108:       9c 40 00 00     l.addi r2,r0,0x0
f000010c:       9c 60 00 00     l.addi r3,r0,0x0
f0000110:       9c 80 00 00     l.addi r4,r0,0x0
f0000114:       9c a0 00 00     l.addi r5,r0,0x0
f0000118:       9c c0 00 00     l.addi r6,r0,0x0
f000011c:       9c e0 00 00     l.addi r7,r0,0x0
f0000120:       9d 00 00 00     l.addi r8,r0,0x0
f0000124:       9d 20 00 00     l.addi r9,r0,0x0
f0000128:       9d 40 00 00     l.addi r10,r0,0x0
f000012c:       9d 60 00 00     l.addi r11,r0,0x0
f0000130:       9d 80 00 00     l.addi r12,r0,0x0
f0000134:       9d a0 00 00     l.addi r13,r0,0x0
f0000138:       9d c0 00 00     l.addi r14,r0,0x0
f000013c:       9d e0 00 00     l.addi r15,r0,0x0
f0000140:       9e 00 00 00     l.addi r16,r0,0x0
f0000144:       9e 20 00 00     l.addi r17,r0,0x0
f0000148:       9e 40 00 00     l.addi r18,r0,0x0
f000014c:       9e 60 00 00     l.addi r19,r0,0x0
f0000150:       9e 80 00 00     l.addi r20,r0,0x0
f0000154:       9e a0 00 00     l.addi r21,r0,0x0
f0000158:       9e c0 00 00     l.addi r22,r0,0x0
f000015c:       9e e0 00 00     l.addi r23,r0,0x0
f0000160:       9f 00 00 00     l.addi r24,r0,0x0
f0000164:       9f 20 00 00     l.addi r25,r0,0x0
f0000168:       9f 40 00 00     l.addi r26,r0,0x0
f000016c:       9f 60 00 00     l.addi r27,r0,0x0
f0000170:       9f 80 00 00     l.addi r28,r0,0x0
f0000174:       9f a0 00 00     l.addi r29,r0,0x0
f0000178:       9f c0 00 00     l.addi r30,r0,0x0
f000017c:       9f e0 00 00     l.addi r31,r0,0x0
f0000180:       18 60 93 00     l.movhi r3,0x9300
f0000184:       a8 63 00 08     l.ori r3,r3,0x8
f0000188:       9c a0 00 00     l.addi r5,r0,0x0
f000018c:       d4 03 28 00     l.sw 0x0(r3),r5
f0000190:       18 60 f0 00     l.movhi r3,0xf000
f0000194:       a8 63 01 a0     l.ori r3,r3,0x1a0
f0000198:       44 00 18 00     l.jr r3
f000019c:       15 00 00 00     l.nop 0x0
Disassembly of section .text:

f00001a0 <_start>:
f00001a0:       04 00 00 0a     l.jal f00001c8 <_init_mc>
f00001a4:       15 00 00 00     l.nop 0x0
f00001a8:       04 00 00 25     l.jal f000023c <_init_mem>
f00001ac:       15 00 00 00     l.nop 0x0
f00001b0:       18 20 00 01     l.movhi r1,0x1
f00001b4:       a8 21 32 c2     l.ori r1,r1,0x32c2
f00001b8:       18 40 f0 00     l.movhi r2,0xf000
f00001bc:       a8 42 02 d8     l.ori r2,r2,0x2d8
f00001c0:       44 00 10 00     l.jr r2
f00001c4:       15 00 00 00     l.nop 0x0

f00001c8 <_init_mc>:
f00001c8:       18 60 93 00     l.movhi r3,0x9300
f00001cc:       a8 63 00 00     l.ori r3,r3,0x0
f00001d0:       9c 83 00 10     l.addi r4,r3,0x10
f00001d4:       18 a0 f0 00     l.movhi r5,0xf000
f00001d8:       b8 a5 00 86     l.srai r5,r5,0x6
f00001dc:       a8 a5 00 25     l.ori r5,r5,0x25
f00001e0:       d4 04 28 00     l.sw 0x0(r4),r5
f00001e4:       9c 83 00 14     l.addi r4,r3,0x14
```

```
f00001e8:       18 a0 19 22     l.movhi r5,0x1922
f00001ec:       a8 a5 00 57     l.ori r5,r5,0x57
f00001f0:       d4 04 28 00     l.sw 0x0(r4),r5
f00001f4:       9c 83 00 08     l.addi r4,r3,0x8
f00001f8:       9c a0 03 ff     l.addi r5,r0,0x3ff
f00001fc:       d4 04 28 00     l.sw 0x0(r4),r5
f0000200:       9c 83 00 00     l.addi r4,r3,0x0
f0000204:       18 a0 0b 00     l.movhi r5,0xb00
f0000208:       a8 a5 03 00     l.ori r5,r5,0x300
f000020c:       d4 04 28 00     l.sw 0x0(r4),r5
f0000210:       9c 83 00 1c     l.addi r4,r3,0x1c
f0000214:       18 a0 00 00     l.movhi r5,0x0
f0000218:       a8 a5 01 03     l.ori r5,r5,0x103
f000021c:       d4 04 28 00     l.sw 0x0(r4),r5
f0000220:       9c 83 00 18     l.addi r4,r3,0x18
f0000224:       18 a0 00 00     l.movhi r5,0x0
f0000228:       b8 a5 00 86     l.srai r5,r5,0x6
f000022c:       a8 a5 04 11     l.ori r5,r5,0x411
f0000230:       d4 04 28 00     l.sw 0x0(r4),r5
f0000234:       44 00 48 00     l.jr r9
f0000238:       15 00 00 00     l.nop 0x0

f000023c <_init_mem>:
f000023c:       9c 60 00 02     l.addi r3,r0,0x2
f0000240:       bc 03 00 00     l.sfeqi r3,0x0
f0000244:       0f ff ff ff     l.bnf f0000240 <_init_mem+0x4>
f0000248:       9c 63 ff ff     l.addi r3,r3,0xffffffff
f000024c:       18 60 f0 00     l.movhi r3,0xf000
f0000250:       a8 63 16 a8     l.ori r3,r3,0x16a8
f0000254:       18 80 00 00     l.movhi r4,0x0
f0000258:       a8 84 00 00     l.ori r4,r4,0x0
f000025c:       18 a0 00 00     l.movhi r5,0x0
f0000260:       a8 a5 00 00     l.ori r5,r5,0x0
f0000264:       e0 a5 20 02     l.sub r5,r5,r4
f0000268:       bc 05 00 00     l.sfeqi r5,0x0
f000026c:       10 00 00 0a     l.bf f0000294 <_init_mem+0x58>
f0000270:       15 00 00 00     l.nop 0x0
f0000274:       84 c3 00 00     l.lwz r6,0x0(r3)
f0000278:       d4 04 30 00     l.sw 0x0(r4),r6
f000027c:       9c 63 00 04     l.addi r3,r3,0x4
f0000280:       9c 84 00 04     l.addi r4,r4,0x4
f0000284:       9c a5 ff fc     l.addi r5,r5,0xfffffffc
f0000288:       bd 45 00 00     l.sfgtsi r5,0x0
f000028c:       13 ff ff fa     l.bf f0000274 <_init_mem+0x38>
f0000290:       15 00 00 00     l.nop 0x0
f0000294:       18 80 00 00     l.movhi r4,0x0
f0000298:       a8 84 20 00     l.ori r4,r4,0x2000
f000029c:       18 a0 00 00     l.movhi r5,0x0
f00002a0:       a8 a5 20 00     l.ori r5,r5,0x2000
f00002a4:       e4 64 28 00     l.sfgeu r4,r5
f00002a8:       10 00 00 07     l.bf f00002c4 <_init_mem+0x88>
f00002ac:       15 00 00 00     l.nop 0x0
f00002b0:       85 03 00 00     l.lwz r8,0x0(r3)
f00002b4:       d4 04 40 00     l.sw 0x0(r4),r8
f00002b8:       9c 63 00 04     l.addi r3,r3,0x4
f00002bc:       0f ff ff fa     l.bnf f00002a4 <_init_mem+0x68>
f00002c0:       9c 84 00 04     l.addi r4,r4,0x4
f00002c4:       9c 60 00 00     l.addi r3,r0,0x0
f00002c8:       9c 80 00 00     l.addi r4,r0,0x0
f00002cc:       44 00 48 00     l.jr r9
f00002d0:       15 00 00 00     l.nop 0x0
f00002d4:       15 00 00 00     l.nop 0x0

f00002d8 <_main>:
f00002d8:       19 e0 00 00     l.movhi r15,0x0
f00002dc:       a9 ef 00 66     l.ori r15,r15,0x66
f00002e0:       19 80 f0 00     l.movhi r12,0xf000
f00002e4:       a9 8c 03 7c     l.ori r12,r12,0x37c
f00002e8:       19 a0 f0 00     l.movhi r13,0xf000
f00002ec:       a9 ad 03 e4     l.ori r13,r13,0x3e4
f00002f0:       19 c0 00 00     l.movhi r14,0x0
f00002f4:       a9 ce 20 00     l.ori r14,r14,0x2000
```

disass.txt 3/3
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
f00002f8:        1b 00 f0 00      l.movhi r24,0xf000
f00002fc:        ab 18 16 a6      l.ori r24,r24,0x16a6
f0000300:        9e 0c 00 00      l.addi r16,r12,0x0
f0000304:        9e 2d 00 00      l.addi r17,r13,0x0
f0000308:        9e 4e 00 00      l.addi r18,r14,0x0
f000030c:        9e 52 ff fc      l.addi r18,r18,0xfffffffc
f0000310:        9d ad ff fe      l.addi r13,r13,0xfffffffe

f0000314 <_next_data>:
f0000314:        9e e0 00 00      l.addi r23,r0,0x0
f0000318:        9e c0 00 02      l.addi r22,r0,0x2
f000031c:        9e a0 00 00      l.addi r21,r0,0x0
f0000320:        e4 11 c0 00      l.sfeq r17,r24
f0000324:        10 00 00 13      l.bf f0000370 <_dump_results>
f0000328:        15 00 00 00      l.nop 0x0
f000032c:        9d ad 00 02      l.addi r13,r13,0x2
f0000330:        9e 2d 00 00      l.addi r17,r13,0x0
f0000334:        9a 91 00 00      l.lhs r20,0x0(r17)
f0000338:        9e 52 00 04      l.addi r18,r18,0x4
f000033c:        e2 00 60 00      l.add r16,r0,r12
f0000340:        9a 70 00 00      l.lhs r19,0x0(r16)

f0000344 <_next_x>:
f0000344:        e2 f3 a3 06      l.mul r23,r19,r20
f0000348:        9e 10 00 02      l.addi r16,r16,0x2
f000034c:        9e 31 00 02      l.addi r17,r17,0x2
f0000350:        9a 70 00 00      l.lhs r19,0x0(r16)
f0000354:        9a 91 00 00      l.lhs r20,0x0(r17)
f0000358:        e2 b5 b8 00      l.add r21,r21,r23
f000035c:        e4 16 78 00      l.sfeq r22,r15
f0000360:        13 ff ff ed      l.bf f0000314 <_next_data>
f0000364:        d4 12 a8 00      l.sw 0x0(r18),r21
f0000368:        03 ff ff f7      l.j f0000344 <_next_x>
f000036c:        9e d6 00 02      l.addi r22,r22,0x2

f0000370 <_dump_results>:
f0000370:        15 00 00 00      l.nop 0x0
f0000374:        03 ff ff ff      l.j f0000370 <_dump_results>
f0000378:        15 00 00 00      l.nop 0x0
```

install_or1k_uclinux.sh

/opt/downloads/studies/or1k/ 1/5 06/28/2004

```
#!/bin/bash

#
# This script is inspired from the ATS script.
#
# Use this to:
# install the openrisc uclinux toolchain.automatically build all the tools and install them.
# Build directories are not removed.
#


#
# Some common variables
#
INSTALL_PREFIX=/opt/or32-uclinux
OK_STR="<font color=#00af00>Test Passed</font>"
FAIL_STR="<font color=#bf0000>Test Failed</font>"
ALL_STABLE=1

#logfiles
GCC295_LOG=/tmp/or1k/logs/gcc-295.log
GCC295_RESULT=/tmp/or1k/logs/gcc-295.status
BINUTILS_LOG=/tmp/or1k/logs/binutils.log
BINUTILS_RESULT=/tmp/or1k/logs/binutils.status
OR1KGCC_LOG=/tmp/or1k/logs/gcc-3.2.3.log
OR1KGCC_RESULT=/tmp/or1k/logs/gcc-3.2.3.status
GDB_LOG=/tmp/or1k/logs/gdb-5.0.log
GDB_RESULT=/tmp/or1k/logs/gdb-5.0.status
OR1KSIM_LOG=/tmp/or1k/logs/or1ksim.log
OR1KSIM_RESULT=/tmp/or1k/logs/or1ksim.status
UCLINUX_LOG=/tmp/or1k/logs/uclinux.log
UCLINUX_RESULT=/tmp/or1k/logs/uclinux.status
UCLIBC_LOG=/tmp/or1k/logs/uclibc.log
UCLIBC_RESULT=/tmp/or1k/logs/uclibc.status
REBUILD_OR1KGCC_LOG=/tmp/or1k/logs/rebuild-gcc-3.2.3.log
REBUILD_OR1KGCC_RESULT=/tmp/or1k/logs/rebuild-gcc-3.2.3.status
#
# Allocate test pool directory
#
echo 'Preparing...'
rm -rfd /tmp/or1k
mkdir /tmp/or1k
mkdir /tmp/or1k/logs
cd or1k
echo 'Copying the needed source tree, this may take a wile'
cp -rfd binutils gcc-3.2.3 gdb-5.0 or1ksim uclinux uclibc ../gcc-2.95.3.tar.gz /tmp/or1k

cd /tmp/or1k
#
# Start with gcc-2.95
#
echo -n 'Building&Installing gcc-2.95.3 (prerequisite)'
gunzip gcc-2.95.3.tar.gz
tar -xf gcc-2.95.3.tar
rm -f gcc-2.95.3.tar
cd gcc-2.95.3
date>$GCC295_LOG 2>&1
uname -a>>$GCC295_LOG 2>&1
nice ./configure --prefix=/opt/gcc-295 --program-transform-name='s/\\(.*\\)/\\1-295/'>>$GCC295_LO
G 2>&1
nice make bootstrap>>$GCC295_LOG 2>&1
nice make install>>$GCC295_LOG 2>&1
BUILD_GCC295_STATUS=$?
if [ $BUILD_GCC295_STATUS = 0 ]; then
	echo '[OK]'
	echo "$OK_STR ('date')" > $GCC295_RESULT
	export PATH=/opt/gcc-295/bin:$PATH
else
	echo '[FAILED]'
	echo "$FAIL_STR ('date')" >  $GCC295_RESULT
	ALL_STABLE=0
	exit
```

install_or1k_uclinux.sh
/opt/downloads/studies/or1k/

2/5
06/28/2004

```
fi
#export
cd ..
#
# Start with binutils
#

mkdir b-b
cd b-b
date > $BINUTILS_LOG 2>&1
uname -a >> $BINUTILS_LOG 2>&1
echo -n 'Building&Installing binutils'
nice env CC=gcc-295 ../binutils/configure --target=or32-uclinux --prefix=$INSTALL_PREFIX >> $BINU
TILS_LOG 2>&1
nice /usr/bin/make all install >> $BINUTILS_LOG 2>&1
BUILD_BINUTILS_STATUS=$?
export PATH=$INSTALL_PREFIX/bin:$PATH
cd ..

#
# Check if binutils was built and installed correctly
#
if [ $BUILD_BINUTILS_STATUS = 0 ]; then
        echo '[OK]'
        echo "$OK_STR ('date')" > $BINUTILS_RESULT
else
        echo '[FAILED]'
        echo "$FAIL_STR ('date')" >  $BINUTILS_RESULT
        ALL_STABLE=0
        exit
fi

#
# Start with gcc
#

mkdir b-gcc
cd b-gcc
date > $OR1KGCC_LOG 2>&1
uname -a >>$OR1KGCC_LOG 2>&1
echo -n 'Building&Installing gcc-3.2.3 for or1k'
nice env CC=gcc-295 ../gcc-3.2.3/configure --target=or32-uclinux \
        --with-gnu-as --with-gnu-ld --verbose \
        --enable-threads --prefix=$INSTALL_PREFIX \
        --local-prefix=$INSTALL_PREFIX/or32-uclinux --enable-languages="c" >>$OR1KGCC_LOG 2>&1
nice /usr/bin/make all install >>$OR1KGCC_LOG 2>&1
BUILD_GCC_STATUS=$?
cd ..

#
# Check if gcc was built and installed correctly
#
if [ $BUILD_GCC_STATUS = 0 ]; then
        echo '[OK]'
        echo "$OK_STR ('date')" > $OR1KGCC_RESULT
else
        echo '[FAILED]'
        echo "$FAIL_STR ('date')">  $OR1KGCC_RESULT
        ALL_STABLE=0
        exit
fi

#
# Start with gdb
#
mkdir b-gdb
cd b-gdb
date > $GDB_LOG 2>&1
uname -a >> $GDB_LOG 2>&1
echo -n 'Building&Installing GDB-5.0: the GNU debugger'
env CC=gcc-295 ../gdb-5.0/configure --target=or32-uclinux --prefix=$INSTALL_PREFIX >> $GDB_LOG 2>
&1
```

```
nice /usr/bin/make all install >> $GDB_LOG 2>&1
BUILD_GDB_STATUS=$?
cd ..

#
# Check if gdb was built and installed correctly
#
echo $BUILD_GDB_STATUS
if [ $BUILD_GDB_STATUS = 0 ]; then
        echo "[OK]"
        echo "$OK_STR ('date')" > $GDB_RESULT
else
        echo "[FAILED]"
        echo "$FAIL_STR ('date')"> $GDB_RESULT
        ALL_STABLE=0
        exit
fi

#
# Start with or1ksim
#
cd or1ksim
date > $OR1KSIM_LOG
uname -a >> $OR1KSIM_LOG
echo -n 'Building&Installing or1ksim: The architectural simulator'
autoconf >>$OR1KSIM_LOG
automake-1.6>>$OR1KSIM_LOG
env CC=gcc-295 ../or1ksim/configure --target=or32-uclinux --prefix=$INSTALL_PREFIX >> $OR1KSIM_LO
G 2>&1
nice /usr/bin/make all install >> $OR1KSIM_LOG 2>&1
BUILD_OR1KSIM_STATUS=$?
cd ..
#
# Check if or1ksim was built and installed correctly
#
if [ $BUILD_OR1KSIM_STATUS = 0 ]; then
        echo "[OK]"
        echo "$OK_STR ('date')" > $OR1KSIM_RESULT
else
        echo "[FAILED]"
        echo "$FAIL_STR ('date')">  $OR1KSIM_RESULT
        ALL_STABLE=0
        exit
fi

#
# Start with uclinux
#
echo  'Building&Installing uClinux'
date > $UCLINUX_LOG
cd uclinux/uClinux-2.0.x
sed -e 's/^LIBGCC/#LIBGCC/; /^#LIBGCC/ a \
LIBGCC = '$INSTALL_PREFIX'\/lib\/gcc-lib\/or32-uclinux\/3.2.3\/libgcc.a' \
< arch/or32/Rules.make > arch/or32/Rules.make.edited
mv arch/or32/Rules.make.edited arch/or32/Rules.make
sed -e 's/^CONFIG_NET/#CONFIG_NET/; /^#CONFIG_NET/ a \
CONFIG_NET=n' < arch/or32/defconfig > arch/or32/defconfig.edited
mv arch/or32/defconfig.edited arch/or32/defconfig
nice make oldconfig >> $UCLINUX_LOG 2>&1
nice make dep >> $UCLINUX_LOG 2>&1
nice /usr/bin/make all >> $UCLINUX_LOG 2>&1
BUILD_UCLINUX_STATUS=$?
echo 'Launching uClinux into the simulator'
touch uart0.rx
echo -e "run 40000000 hush\nq" | or32-uclinux-sim -f sim.cfg linux -i > /dev/null
cp uart0.tx /tmp/or1k/logs/or1ksim_uclinux.txt
cd /tmp/or1k
#
# Check if uclinux was built and installed correctly
#
if [ $BUILD_UCLINUX_STATUS = 0 ]; then
        echo "Compilation of uClinux...OK"
```

```
        echo "$OK_STR ('date')" > $UCLINUX_RESULT
else
        echo "Compilation of uClinux...FAILED"
        echo "$FAIL_STR ('date')"> $UCLINUX_RESULT
        ALL_STABLE=0
        exit
fi
if [ 'grep Executing /tmp/or1k/logs/or1ksim_uclinux.txt | wc -l' -gt 0 ]; then
        echo "Simulation of uClinux...OK"
        echo "$OK_STR ('date')" > /tmp/or1k/logs/or1ksim_uclinux.status
else
        echo "Simulation of uClinux...FAILED"
        echo "$FAIL_STR ('date')"> /tmp/or1k/logs/or1ksim_uclinux.status
        ALL_STABLE=0
        exit
fi

#
# Start with uclibc
#
echo 'Building&Installing Uclibc'
date > $UCLIBC_LOG 2>&1
cd uclibc
date > $UCLIBC_LOG 2>&1
uname -a >> $UCLIBC_LOG 2>&1
ln -s ./extra/Configs/Config.cross.or32.uclinux Config
sed -e 's/^KERNEL_SOURCE/#KERNEL_SOURCE/; /^#KERNEL_SOURCE/ a \
KERNEL_SOURCE = ''pwd''\/..\/uclinux\/uClinux-2.0.x' < Config > Config.edited
sed -e 's/^DEVEL_PREFIX/#DEVEL_PREFIX/; /^#DEVEL_PREFIX/ a \
DEVEL_PREFIX = '$INSTALL_PREFIX < Config.edited > Config
rm Config.edited
nice /usr/bin/make all install >> $UCLIBC_LOG 2>&1
BUILD_UCLIBC_STATUS=$?

pushd .>/dev/null
cd $INSTALL_PREFIX/bin
rm -f ar as cc cpp gasp gcc ld nm objcopy objdump ranlib size strings strip jar grepjar
popd>/dev/null
cd ../b-gcc
date > $REBUILD_OR1KGCC_LOG 2>&1
uname -a >> $REBUILD_OR1KGCC_LOG 2>&1
echo 'Rebuild&Reinstall of gcc-3.2.3 for or1k to use uClibc'
nice env CC=gcc-295 ../gcc-3.2.3/configure --target=or32-uclinux \
         --with-gnu-as --with-gnu-ld --verbose \
         --enable-threads --prefix=$INSTALL_PREFIX \
         --local-prefix=$INSTALL_PREFIX/or32-uclinux --enable-languages="c" >> $REBUILD_OR1KGCC_LO
G 2>&1
nice make all install >> $REBUILD_OR1KGCC_LOG 2>&1
REBUILD_GCC_STATUS=$?
cd ..

#
# Check if uclibc was built and installed correctly
#
if [ $BUILD_UCLIBC_STATUS = 0 ]; then
         echo "Uclibc build&install...[OK]"
         echo "$OK_STR ('date')" > $UCLIBC_RESULT
else
         echo "Uclibc build&install...FAILED"
         echo "$FAIL_STR ('date')"> $UCLIBC_RESULT
         ALL_STABLE=0
         exit
fi
if [ $REBUILD_GCC_STATUS = 0 ]; then
         echo "gcc-3.2.3 re build&reinstall...OK"
         echo "$OK_STR ('date')" > $REBUILD_OR1KGCC_RESULT
else
         echo "gcc-3.2.3 re build&reinstall...FAILED"
         echo "$FAIL_STR ('date')"> $REBUILD_OR1KGCC_RESULT
         ALL_STABLE=0
         exit
fi
```

install_or1k_uclinux.sh	5/5
/opt/downloads/studies/or1k/	06/28/2004

install_or1k_uclinux.sh	**5/5**
/opt/downloads/studies/or1k/	06/28/2004

sim.cfg 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
section memory
        type = unknown
        nmemories = 2
        device 0
                name = "FLASH"
                ce = 0
                baseaddr = 0xf0000000
                size = 0x00800000
                delayr = 1
                delayw = -1
        enddevice
        device 1
                name = "RAM"
                ce = 1
                baseaddr = 0x00000000
                size = 0x00800000
                delayr = 1
                delayw = 1
        enddevice
end
section mc
        enabled = 1
        baseaddr = 0x93000000
        POC = 0x00000008
end
section sim
        verbose = 1
end
```

dump2wintxt.pl 1/1
~/work/nezhate/or1k/asm/fir1/ 06/28/2004

```
#!/usr/bin/perl

$dump_file=$ARGV[0];
$win_file=$ARGV[1];

open(DUMP,$dump_file);
@dump_file=<DUMP>;
close(DUMP);

chop(@dump_file);
$i=0;


open(FOUT,">$win_file");
foreach $line(@dump_file){
    $line =~ /.*\:\s(.*)/;
#    print $1."\n";
    @line=split(" ",$1);
    for($i=0;$i <= 3;$i++){
        $word[$i]=join("",@line[0+$i*4..3+$i*4]);
        print FOUT "$word[$i]\n";
    }
}
close(FOUT);
system "unix2dos $win_file";
```