Titre: Title:	Conception d'un processeur embarqué de faible complexité dédié à une plate-forme SoC pour concevoir des processeurs réseaux
Auteur: Author:	Hany Ghattas
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ghattas, H. (2003). Conception d'un processeur embarqué de faible complexité dédié à une plate-forme SoC pour concevoir des processeurs réseaux [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7123/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7123/
Directeurs de recherche: Advisors:	Yvon Savaria
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UN PROCESSEUR EMBARQUÉ DE FAIBLE COMPLEXITÉ DÉDIÉ À UNE PLATE-FORME SoC POUR CONCEVOIR DES PROCESSEURS RÉSEAUX

HANY GHATTAS
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
JUILLET 2003

Your file Votre référence
ISBN: 0-612-86400-6
Our file Notre référence
ISBN: 0-612-86400-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CONCEPTION D'UN PROCESSEUR EMBARQUÉ DE FAIBLE COMPLEXITÉ DÉDIÉ À UNE PLATE-FORME SoC POUR CONCEVOIR DES PROCESSEURS RÉSEAUX

présenté par : GHATTAS Hany

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. AUDET Yves, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BOIS Guy, Ph.D., membre

REMERCIEMENTS

J'aimerais tout d'abord remercier les organismes subventionnaires qui m'ont soutenu financièrement au cours de mes études de maîtrise et de la réalisation du projet décrit dans le présent mémoire. Ceux-ci sont : la compagnie Gennum Corporation ainsi que le réseau de centres d'excellence sur les dispositifs, circuits et systèmes microélectroniques, financé par le gouvernement fédéral et l'industrie au Canada et Micronet.

J'aimerais remercier les gens qui m'ont guidé tout au long de ce projet de recherche, soient mon directeur de recherche, le professeur Yvon Savaria de l'École Polytechnique de Montréal, ainsi que Messieurs Jean-Marc Tremblay et Arnold Veenstra, ingénieurs chez Gennum Corporation. Également, mes collègues de travail qui m'ont éclairé sous différents sujets, soient Mme Maria Mbaye, M. Sébastien Regimbal, M. Jean Pepga-Bissou étudiants en maîtrise à l'École Polytechnique de Montréal.

J'aimerais aussi remercier les gens avec qui j'ai travaillé ou que j'ai côtoyés tout au long de mes travaux. Ceux-ci sont : M. Tien Bui, M. Alexandre Chureau et M. Kevin Peterson.

Je voudrais enfin remercier ma mère Marie ainsi que mon père Victor, qui m'ont supporté pendant toute la durée de mes études.

RÉSUMÉ

Les composants pour réaliser des systèmes de traitement de flot réseau («Network Processor ») sont des dispositifs programmables effectuant le traitement, dans des dispositifs de communication de données, des unités de données de protocole, à très haute vitesse. Ils ont émergé suite à la demande croissante des services améliorés pour une prochaine génération. Cette recherche fait partie d'un effort pour mettre en application une plate-forme « SoC » qui pourrait supporter des paquets vidéo numériques de très haute vitesse avec une basse latence. Cette plate-forme flexible est basée sur un processeur embarqué fait sur mesure. Nous avons également développé un assembleur qui sera utile pour différentes applications. Des algorithmes simples de conversion de protocole ont été codés pour ce processeur. Cette plate-forme « SoC » permet au concepteur de système de remplacer les puces intégrées à fonctionnalités fixes ainsi que les processeurs RISC, par des dispositifs intelligents et programmables qui maintiennent une très haute vitesse.

Ce mémoire présente une brève étude de plusieurs processeurs réseaux existant sur le marché. Il propose une architecture de plate-forme « SoC » capable de faire la conversion de protocoles, ainsi que plusieurs applications de télécommunication telle que la classification de paquets. Nous avons mis en œuvre un processeur embarqué fait sur mesure qui sera intégré dans cette plate-forme, utile pour la manipulation de paquets. Ce mémoire compare plusieurs versions de notre processeur embarqué avec le processeur ARM7, un processeur populaire dont le noyau est disponible sur le marché. Il démontre quelques avantages d'un processeur embarqué au cœur d'une plate-forme « SoC », consacrée à la transmission de paquets de vidéo numérique. Ce processeur, fait sur mesure, offre un rendement plus élevé, et pourrait être facilement adapté à nos besoins; cependant, nous n'avons pas réussi à atteindre la densité du processeur ARM7 à cause de la bibliothèque de cellules disponibles et de la méthodologie de conception physique choisie.

ABSTRACT

Network processing system components are programmable devices performing wire-speed processing of ***Protocol Data Units*** in data communication devices. They emerged as a direct result of the growing demand for enhanced, flexible next generation communication services. This research is part of an effort to implement a SOC platform that could support high throughput and low latency real time video streaming. This flexible platform is based on a custom embedded processor for which we also developed an application specific assembler. Simple protocol conversion algorithms were coded for this processor. This SoC platform allows system designer to replace fixed-functionality ASICs and RISC CPUs in the critical path with intelligent, programmable devices that maintain wire speed.

This Master's thesis presents a brief study of several Network processors that are already in the market. It proposes a SoC platform architecture capable of doing protocols conversion among other telecommunication applications such as classification. We have implemented a custom embedded processor that will be integrated in this SoC platform for packets manipulation. This thesis compares several versions of our embedded processor with the ***ARM7***, a popular core processor available in the market. It demonstrates some benefits of an embedded processor in a SOC platform dedicated to video streaming packets. The custom processor offers a higher performance, and could be easily adapted to our needs; however, we could not approach the density of the ***ARM7*** with the available cell library and physical design flow.

TABLE DES MATIÈRES

REMERCIEMENTS ..... **iv**

RÉSUMÉ ..... **v**

ABSTRACT ..... **vi**

LISTE DES FIGURES ..... **xi**

LISTE DES TABLEAUX ..... **xiii**

LISTE DES ACRONYMES ..... **xiv**

INTRODUCTION **1**

CHAPITRE I **6**

Revue critique de la littérature **6**

1.1 Évolution des processeurs réseau 6

1.1.1 Processeurs réseau Ethernet 6

1.1.2 Processeurs réseau Gigabit Ethernet, OC-12 et OC-48 8

1.1.2.1 Commutateurs de réseau basés sur des processeurs RISC 8

1.1.2.2 Commutateurs réseau basés sur des circuits intégrés ASIC 9

1.1.2.3 Commutateurs réseau basés sur des processeurs RISC améliorés 10

1.1.2.4 Commutateurs réseau basés sur des approches hybrides 12

1.1.2.5 Architecture de processeurs intermédiaires 12

1.2 Processeur réseau 13

1.2.1 Définition d'un processeur réseau 13

1.2.2 Caractéristiques d'un processeur réseau 14

1.2.3 Le routage 15

1.3 Le marché des processeurs réseau OC-192 16

1.3.1 Alchemy (Au1000) 17

1.3.1.1 Architecture interne 17

1.3.1.2 Programmabilité 17
1.3.1.3 Implantation 18
1.3.2 IBM (PowerNP) 18
1.3.2.1 Architecture interne 19
1.3.2.2 Programmabilité 19
1.3.2.3 Implantation 19
1.3.3 Intel (IXP1200) 20
1.3.3.1 Architecture interne 21
1.3.3.2 Programmabilité 22
1.3.3.3 Implantation 22
1.3.4 Motorola (C-5 DCP) 23
1.3.4.1 Architecture interne 23
1.3.4.2 Programmabilité 23
1.3.4.3 Implantation 24
1.3.5 Autres processeurs réseau 24
1.3.6 Synthèse de trois principaux processeurs réseau 25

CHAPITRE II 27

Architecture globale de la plate-forme SoC 27
2.1 Architecture interne 27
2.1.1 Première version du modèle exécutable et processus de conversion 27
2.1.2 Évolution de l'architecture 31
2.1.3 Description des rôles des différents modules 31
2.1.3.1 Les interfaces 31
2.1.3.2 Le gestionnaire de mémoire 33
2.1.3.3 Les mémoires 33
2.1.3.4 Le contrôleur 33
2.1.3.5 Le coprocesseur d'identification des protocoles 34
2.1.3.6 Le coprocesseur de vérification / conversion d'adresses 34
2.1.3.7 Le coprocesseur d'assemblage des paquets 34

2.1.3.8 Le coprocesseur d'intégrité des données 35
2.1.3.9 Le « Core Connect Bus » d'IBM 35
2.1.3.10 Le processeur ARM 36
2.2 Caractéristiques générales de la première version 37
2.2.1 Évaluation des communications 37
2.2.1.1 Évaluation de la largeur de bande utile disponible d'un réseau : analyse et comparaisons 38
2.2.1.2 Évaluation de la largeur de bande nécessaire pour transmettre de la vidéo numérique de 360 Mbps 40
2.2.1.3 Évaluation du nombre d'opérations disponibles par paquet 41
2.2.1.4 Évaluation de la largeur de bande des bus de communication internes du processeur réseau 42
2.2.2 Protocoles supportés 44
2.2.2.1 Le modèle OSI 44
2.2.2.2 IEEE 802.3 (Gigabit Ethernet) 46
2.2.2.3 IEEE 1394 (Firewire) 46

CHAPITRE III 48

Conception d'un processeur embarqué dédié 48
3.1 Les entrées/sorties du processeur embarqué 50
3.2 Description des signaux d'Entrées/Sorties 50
3.3 Description des registres 53
3.4 Détails des opérations de lecture et d'écriture 54
3.4.1 Interface entre le GF et le processeur ARM7 54
3.4.2 Interface entre le GF et le coprocesseur de conversion d'adresses AC 55
3.4.3 Interface entre le GF et la mémoire principale 56
3.4.4 Interface entre le GF et la mémoire temporaire 57
3.4.5 Interface entre le GF et le coprocesseur d'assemblage de paquets PA 58
3.4.6 Interface entre le GF et le contrôleur 59
3.5 Détails du Reset 60

3.6 Détails des opérations arithmétiques, logiques, transfert et de branchement ... 60
3.6.1 Opérations arithmétiques et logiques ... 60
3.6.2 Opérations de décalage ... 61
3.6.3 Opérations de transfert ... 61
3.6.4 Opérations de saut ... 62
3.6.5 Instructions spéciales ... 62
3.7 Méthodologie de vérification ... 63
3.7.1 Cas de test ... 67
3.7.2 Cas de test du processeur embarqué ... 68
3.7.3 Un condensé des fonctionnalités du GF ... 72

CHAPITRE IV ... 79

Implantations et résultats ... 79
4.1 Assembleur ... 79
4.2 Architecture interne du processeur embarqué ... 81
4.3 Résultats de synthèse sur un FPGA ... 83
4.4 Résultats de synthèse sur un circuit intégré ASIC ... 84
4.5 Performance de notre processeur embarqué ... 84
4.6 Discussions ... 89
4.6.1 Les processeurs configurables ... 89
4.6.2 Le processeur logiciel MicroBlaze offert par la compagnie Xilinx ... 90

CONCLUSION ... 92

RÉFÉRENCES ... 96

ANNEXE A.1 ... 100

Détails des opérations du jeu d'instructions ... 100

LISTE DES FIGURES

Figure 1.1 Architecture traditionnelle basée sur un processeur central 7
Figure 1.2 Vitesse des réseaux et demande de bande passante comparées à la progression des performances de la technologie d'intégration 9
Figure 1.3 L'architecture interne du Au1000 [3] 18
Figure 1.4 Schéma bloc de l'architecture interne d'un processeur réseau IBM-PowerNP [20] 20
Figure 1.5 Schéma bloc de l'architecture interne du Intel-IXP1200 [24] 22
Figure 2.1 Architecture de la première version 30
Figure 2.2 Architecture globale 31
Figure 2.3 Largeur de bande (LB) max. du bus de communication 42
Figure 2.4 Couches du modèle OSI 45
Figure 3.1 Diagramme Bloc de haut-niveau du GF 50
Figure 3.2 Registre d'états (SREG) 53
Figure 3.3 Diagramme temporel de l'interface GF-ARM 55
Figure 3.4 Diagramme temporel de l'interface GF-AC 56
Figure 3.5 Diagramme temporel de l'interface GF-Mémoire principale 57
Figure 3.6 Diagramme temporel de l'interface GF-Mémoire temporaire en mode lecture 58
Figure 3.7 Diagramme temporel de l'interface GF-Mémoire temporaire en mode écriture 58
Figure 3.8 Diagramme temporel de l'interface GF-PA 59
Figure 3.9 Diagramme temporel de l'interface GF-CTR 60
Figure 3.10 : Méthodologie de vérification 64
Figure 3.11 Schéma du banc d'essai du GF 73
Figure 4.1 Un exemple de code d'assembleur 81
Figure 4.2 Un exemple de code binaire 81
Figure 4.3 Schéma bloc de haut niveau de l'architecture du processeur embarqué 83

Figure 4.3 Architecture interne du processor logiciel *MicroBlaze* de Xilinx [50] 91

LISTE DES TABLEAUX

Tableau 1.1 Synthèse et comparaison des trois principaux processeurs réseau 26
Tableau 2.1 Comparaison entre le bus ARM AMBA 2.0 et le bus IBM CoreConnect.... 36
Tableau 2.2 Largeurs de bande utilisées pour des paquets de tailles maximales 39
Tableau 2.3 Largeurs de bande utilisées pour des paquets de tailles minimales permettant la transmission de vidéo numérique de 360 Mbps..... 41
Tableau 2.4 Largeurs de bande requises par les différents éléments de l'architecture..... 44
Tableau 2.5 Description du paquet *Ethernet*..... 46
Tableau 2.6 Description du paquet *Firewire* 47
Tableau 3.1 Description du registre du statut 53
Tableau 3.2 Tableau illustrant la sélection d'un module par le GF 54
Tableau 3.3 Le jeu d'instructions utilisé par l'engin de formatage 63
Tableau 3.4 Détails du plan de vérification 65
Tableau 3.5 Description du partitionnement par aspect..... 66
Tableau 3.6 GF - Tests par fonctionnalité 69
Tableau 3.7 GF - Test par interface 71
Tableau 3.8 Résumé des fonctionnalités du GF..... 73
Tableau 4.1 Jeu d'instructions du GF contre celui d'un processeur RISC 85
Tableau 4.2 Performances des trois versions du processeur embarqué 87
Tableau 4.3 Performances de la version définitive du processeur embarqué contre *ARM7TDMI*..... 87

LISTE DES ACRONYMES

ACL	*Access Control List*
ASIC	*Application Specific Integrated Circuit*
ASSP	*Application Specific Standard Product*
ATM	*Asynchronous Transfer Mode*
CPU	*Central Processing Unit*
CMC	*Canadian Microelectronics Corporation*
CRC	*Cyclic Redundancy Check*
DA	*Destination Address*
DLL	*Data Link Layer*
DSP	*Digital Signal Processor*
DUT	*Design Under Test*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
FCS	*Frame Check Sequence*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*
Gbps	*Giga bit per second*
GF	*General Formatter*
GRM	*Groupe de Recherche en Microéléctronique*
HDLC	*High-level Data Link Control*
IP	*Internet Protocol*
LAN	*Local Area Network*
LUT	*Look-Up Table*
Mbps	*Mega bit per second*
MIPS	*Millions of Instruction Per Second*
NL	*Network Layer*

OSI	*Open System Interconnection*
PCI	*Peripheral Connection Interface*
RISC	*Reduced Instruction Set of Complexity*
RTL	*Register Transfer Language*
RTP	*Real-Time Transport Protocol*
SoC	*System on a Chip*
SONET	*Synchronous Optical Network*
TCAM	*Ternary Content Addressable Memory*
TCP	*Transmission Control Protocol*
TL	*Transport Layer*
UART	*Universal Asynchronous Receiver and Transmitter*
UDP	*User Datagram Protocol*
VLSI	*Very Large Scale Integrated Circuit*
WAN	*Wide Area Network*

INTRODUCTION

Le monde des télécommunications, entraîné par l'évolution rapide de la microélectronique, a connu de nombreuses mutations ces dernières décennies. Le marché se caractérise par une compétition intense et une course effrénée vers de plus grandes bandes passantes. La demande en bande passante est particulièrement pressante pour les réseaux, qui ont vu naître de nombreux protocoles. Ainsi, depuis le début des années 90, de nombreuses recherches ont été réalisées dans le but de développer de nouveaux protocoles de communication capables de supporter des débits de plus en plus grands. Ces protocoles, permettant un transfert de données pouvant s'élever à plus de 10 Gbps (*Giga bit per second*), sont en continuelle évolution, ce qui rend leur communication sur un même réseau extrêmement difficile. De plus, cette constante innovation dans le domaine de la communication, entraîne une diminution de la durée de vie des produits qui se trouvent dépassés dès qu'ils ne sont pas capables d'assurer une certaine vitesse de transmission. Par conséquent, sous peine de devoir renouveler des équipements entiers, il est devenu indispensable de faire communiquer des équipements supportant des protocoles hétérogènes entre eux. Pour ce faire, l'industrie des télécommunications a fait appel à des interfaces réseaux spécialisées : les convertisseurs de protocoles.

Cette large variété de protocoles hétérogènes crée des défis difficiles à relever pour les intégrateurs de réseaux. Cela implique la nécessité de permettre la communication entre équipements vieux et nouveaux. Ceci est accompli par les convertisseurs de protocoles. Ces derniers sont, soient implantés dans des circuits intégrés ASIC (*Application Specific Integrated Circuit*), soient basés sur des processeurs réseaux [1]. Dans le premier cas, les désavantages sont non seulement les coûts non-récurrents élevés et le délai d'arrivée sur le marché, mais également le manque de flexibilité. Quant aux processeurs embarqués standards, s'ils sont plus flexibles, ils sont néanmoins dérivés de processeurs RISC (*Reduced Instruction Set of Complexity*), et utilisent des jeux d'instructions qui ne sont

pas dédiés aux opérations de manipulation de paquets, notamment la conversion de protocoles. Ainsi, leur utilisation pourrait être non-optimisée.

Le projet de Convertisseur de Protocoles est un travail de recherche financé par l'entreprise GENNUM Corporation. C'est une entreprise canadienne implantée à Burlington en Ontario, qui conçoit, réalise et commercialise des composants électroniques, essentiellement des circuits intégrés à base de silicium et également des circuits hybrides pour des applications spécialisées. GENNUM subventionne ce projet afin d'acquérir en retour une base de données sur les choix technologiques et architecturaux, les problématiques rencontrées et les solutions envisagées. Ces informations serviront de base de travail pour la commercialisation future de convertisseurs de protocoles. Un premier mémoire de Maîtrise [25], dans le cadre de ce projet, a été déjà déposé par monsieur Lepage. Il a proposé une méthode d'identification par apprentissage applicable à la classification des paquets de haut niveau.

Le but de ce projet est de réaliser une entité qui assurera la conversion d'un protocole A en un protocole B. Ainsi, un flux de données (dans notre cas des données vidéo), ayant été transmis à l'aide d'un ensemble de protocoles, pourra être traité par une structure utilisant un ensemble de protocoles différents. Ce convertisseur doit être expressivement générique, pour pouvoir traiter le plus grand nombre de protocoles possibles, flexible [2], afin que le processus de conversion ne soit pas unique mais qu'il s'adapte au protocole traité, et enfin, réutilisable, pour pouvoir le faire évoluer en fonction de l'évolution de la technologie propre aux réseaux, notamment les nouveaux protocoles.

Ce convertisseur de protocoles trouverait son utilité dans un studio de montage vidéo; en effet, ce type de studio possède des outils de traitement de l'image très coûteux dont la conception ancienne n'autorise qu'un flot de données entrant supporté par un ensemble de protocoles défini. Par conséquent, il faut convertir les protocoles des données vidéo

reçues de façon à les rendre compatibles avec les protocoles attendus par les différents outils du studio, tout en conservant une qualité d'image optimale.

Dans un premier temps, le but est d'obtenir un modèle exécutable qui puisse démontrer le bon fonctionnement des différents modules du convertisseur en réalisant la conversion d'un protocole A (Ethernet ou Firewire) en un protocole B (Firewire ou Ethernet), deux protocoles de la couche liaison de données que nous décrirons brièvement ultérieurement. Par la suite, l'ensemble du convertisseur sera modifié de façon à devenir synthétisable et à supporter trois couches du modèle OSI (*Open System Interconnection*), qui sont la couche liaison de données, la couche réseau et la couche transport. Nous pourrons procéder à l'implantation de notre convertisseur sur une plate-forme ARM-FPGA.

La plate-forme du convertisseur de protocoles est basée sur un ou plusieurs processeurs embarqués, considérés comme des engins de formatage. Ces derniers réalisent l'essentiel de la conversion. Ils peuvent être configurés et programmés pour exécuter le programme de conversion. Ils confèrent un haut niveau de flexibilité. Chaque engin de formatage est en charge d'un paquet et travaille en parallèle avec les autres engins de formatage. Leur jeu d'instructions est dédié aux opérations de conversion.

Cette phase de recherche devrait nous conduire à réaliser des convertisseurs de protocoles supportant plus de protocoles, avec un nombre optimisé d'engins de formatage. Un processeur embarqué conventionnel, le ***ARM7***, est chargé de la gestion d'exceptions. Quant à la conception physique, la première implémentation vise une plate-forme FPGA (*Field Programmable Gate Array*), un circuit intégré ASIC devrait suivre, lorsque l'architecture sera arrivée à maturité.

Notre proposition consiste en une nouvelle solution matérielle pour réaliser certaines applications de télécommunications, en l'occurrence, la conversion de protocoles. L'architecture proposée permettra également le traitement d'exceptions ou d'opérations

complexes à l'aide d'un processeur embarqué conçu sur mesure pour la manipulation des données de type vidéo numérique. Ce mémoire traitera principalement de la nécessité d'un processeur embarqué et du jeu d'instructions dédié aux traitements de paquets.

Dans un monde où l'évolution des développements technologiques se poursuit à un rythme accéléré, les concepteurs de circuits intégrés VLSI (*Very Large Scale Integrated Circuit*) doivent affronter un défi constant de performance. Ainsi, avec la forte croissance de la demande pour des équipements réseau à très grande largeur de bande, le processeur ou noyau principal des plates-formes réseau est bien loin de la performance désirée, malgré le succès exceptionnel de l'architecture des processeurs RISC. Les divers problèmes de flexibilité, de programmabilité et de temps de mise en marché ont poussé jusqu'à présent les spécialistes vers la recherche de solutions, visant à offrir de nouvelles architectures de processeurs réseau qui répondent aux besoins du marché.

Ce mémoire expose les travaux qui ont été effectués dans le cadre du développement d'une plate-forme SoC (*System on a Chip*) dédiée à la réalisation de processeurs réseau. Elle est basée sur un processeur embarqué dont le jeu d'instructions est spécifique pour le traitement de paquets à très haut débit.

Ainsi, le premier chapitre de ce mémoire est essentiellement une revue des réalisations commerciales et de la littérature en ce qui a trait au domaine des processeurs réseau. Il existe plusieurs modèles de processeurs réseau disponibles commercialement. En fait, ce chapitre présente un survol historique sur l'évolution des processeurs réseau des années 90 jusqu'à présent. Puis, une étude sur quatre processeurs réseau, fabriqués par des entreprises bien connues, sera présentée. Finalement, une foire aux processeurs réseau avec leurs pointeurs sera donnée au cas où le lecteur voudrait examiner plus de processeurs réseau.

Le deuxième chapitre présentera l'ensemble des critères qui nous ont permis d'élaborer le cahier de charges de notre plate-forme qui servira comme véhicule pour le convertisseur de protocoles. Ce sont ces attentes spécifiques qui, ajoutées aux spécifications propres aux protocoles et aux impératifs de la communication de signaux vidéo, nous ont permis d'aboutir à l'architecture interne de notre plate-forme que nous présenterons dans ce même chapitre. Cette architecture se veut un modèle exécutable capable d'effectuer la conversion d'un protocole A en un protocole B de la couche liaison de données.

Le troisième chapitre présentera le modèle du processeur embarqué qui a été développé pour devenir le cœur de notre plate-forme SoC. Ainsi, nous expliquerons d'abord le jeu d'instructions nécessaire pour accomplir une application de conversion de protocoles. Puis, les parties importantes du processeur, notamment les différentes interfaces nécessaires à la communication avec les divers coprocesseurs de la plate-forme, seront décrites. Finalement, nous discuterons de notre plan de vérification, pour assurer un minimum de niveau de confiance nécessaire lors de la réalisation de ce modèle de processeur.

Le quatrième et dernier chapitre présentera le travail qui a été réalisé pour implanter notre processeur embarqué. Nous commencerons par présenter l'assembleur qui a été développé par un chercheur stagiaire, monsieur Armin Schneider. Puis, nous illustrerons l'architecture interne de notre processeur qui sera suivie des résultats de synthèse sur un FPGA et sur un circuit intégré ASIC en technologie 0,35 µm. Finalement, une comparaison entre notre processeur embarqué et le processeur ***ARM7***, déjà disponible sur le marché, sera détaillée.

CHAPITRE I

Revue critique de la littérature

Dans ce chapitre, nous ferons un bref historique des différentes architectures utilisées de nos jours, les limites rencontrées par ces machines, ainsi que les besoins urgents d'une nouvelle génération de processeurs réseau.

Ensuite, nous aborderons les détails de diverses architectures actuelles, que les compagnies conçoivent dans l'espoir de surpasser les limites rencontrées par les modèles précédents. Enfin, nous présenterons les différents processeurs réseau actuellement sur le marché, la vision des spécialistes et les perspectives des fabricants pour les prochaines années à venir.

1.1 Évolution des processeurs réseau

Les équipements réseau étaient généralement conçus à partir d'une combinaison de CPU (*Central Processing Unit*), de circuiterie logique, de produits standards d'application spécifique ASSP (*Application Specific Standard Product*), de contrôleur d'interface et de trans-récepteur. Tous ces composants exigeaient un logiciel assez dense, qui devenait de plus en plus complexe, quand il fallait ajouter ou modifier certaines fonctionnalités du réseau.

1.1.1 Processeurs réseau Ethernet

Jusqu'à la fin des années 90, la majorité des commutateurs fonctionnant à des vitesses comparables à Ethernet (10 Mbps (*Mega bit per second*)) étaient basés sur une

architecture similaire à un ordinateur personnel. Un CPU performait toutes les fonctions de routage de paquets selon divers protocoles de communication.

Le CPU recevait du système d'exploitation des instructions qui s'exécutaient dans la mémoire vive à partir des instructions de base qui étaient implantées dans la mémoire cache. L'avantage de ce type d'architecture résidait dans le fait, que toutes les instructions étaient contenues dans le programme, l'ajout de nouvelles fonctionnalités était facile via une modification dans le programme du système, sans toucher à l'architecture du processeur de base. Avec la demande élevée de largeur de bande et de nouvelles fonctions, cette architecture est devenue trop dispendieuse à maintenir. À mesure que la vitesse approche du 1 Gbps, les microprocesseurs traditionnels sont devenus surchargés et occasionnaient ainsi des délais considérables dans le réseau. De plus, les modèles que nous rencontrons sur le marché sont capables de supporter les signaux OC-3 à une vitesse pouvant atteindre 155 Mbps et dans les pires cas, ils pourraient être utilisés pour les signaux OC-12 à une vitesse atteignant 622 Mbps. Un exemple parfait de cette approche est le produit ***CISCO 2500*** [10] disponible dans diverses configurations selon l'application désirée. Nous pouvons disposer d'un simple port ou de multiples ports série de type « AUI » ou « Token Ring ». Chacune de ces architectures, utilisant un CPU, est connectée aux interfaces physiques et aux mémoires à travers un bus système. Ainsi, le dispositif est capable de traiter n'importe quelle fonction pour le flot de données entrant, à mesure que nous mettons à jour le logiciel. La figure 1.1 illustre l'architecture traditionnelle basée sur un processeur central.

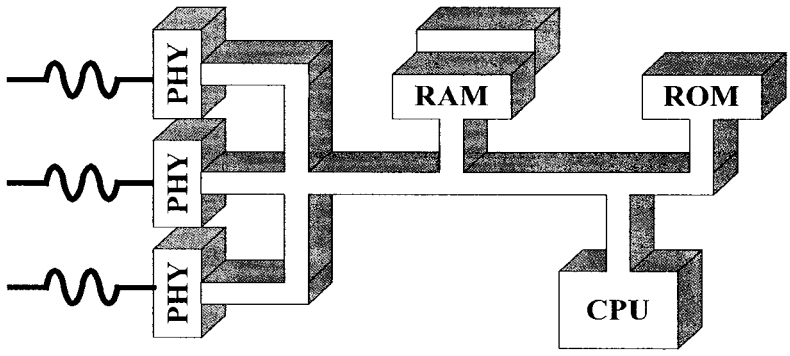


Figure 1.1 Architecture traditionnelle basée sur un processeur central

1.1.2 Processeurs réseau Gigabit Ethernet, OC-12 et OC-48

Récemment, les commutateurs qui fonctionnent à des vitesses comparables aux Gigabit Ethernet (1000 Mbps), OC-12 (622 Mbps) et OC-48 (2488 Mbps), ont pu voir le jour grâce à l'adoption d'une architecture basée sur des processeurs RISC, des circuits intégrés à très grande échelle, des RISC améliorés et des modèles hybrides.

1.1.2.1 Commutateurs de réseau basés sur des processeurs RISC

Pour un processeur RISC, il faut plusieurs instructions pour exécuter une tâche simple. Le nombre d'instructions augmente avec la complexité de la tâche, ce qui augmente le temps d'exécution puisque chaque instruction prend un coup d'horloge.

En raison du nombre élevé d'instructions et du temps d'exécution des tâches, le processeur réseau basé sur un modèle RISC convient plus pour des commutateurs se limitant aux vitesses de OC-48. Si le nombre d'instructions est moins élevé, le processeur RISC fournit une bonne performance, une flexibilité et un haut degré de programmabilité. Néanmoins, un tel commutateur demeure lent en raison des décisions qui sont prises à l'intérieur du logiciel.

Les processeurs RISC sont souvent utilisés dans une configuration parallèle pour augmenter la vitesse de fonctionnement du commutateur. Cette architecture a désormais, ses limites puisque le nombre de processeurs RISC pouvant être incorporé sur une même puce augmente sa complexité et ses dimensions. De plus, une telle architecture permet aux commutateurs de supporter une grande variété de protocoles et de spécifications, bien qu'ils ne s'adaptent pas facilement aux ajouts de fonctionnalités qui exigent des bandes passantes spécifiques. Par exemple, le rajout d'une fonction de contrôle d'accès ACL (*Access Control List)* sur une architecture à processeur RISC pourrait occasionner un niveau d'utilisation maximal et engendrer ainsi des perturbations au niveau du trafic.

Les processeurs RISC ont beaucoup de flexibilité puisqu'il suffirait de changer le programme à l'intérieur du processeur pour modifier ou ajouter des nouvelles fonctionnalités. Mais, ils n'ont pas la même capacité de traitement que les circuits intégrés dédiés, puisque leur architecture est de type général, c'est-à-dire qu'ils sont d'usage général et ne sont pas spécifiquement dédiés au domaine des réseaux.

La performance des processeurs RISC s'est améliorée depuis des années, en accord avec la loi de Moore qui conduit à une augmentation de la densité des transistors à tous les 18 mois. Le diagramme de Moore, montré à la figure 1.2, prévoit des limites de réduction d'échelle par rapport aux exigences des réseaux.

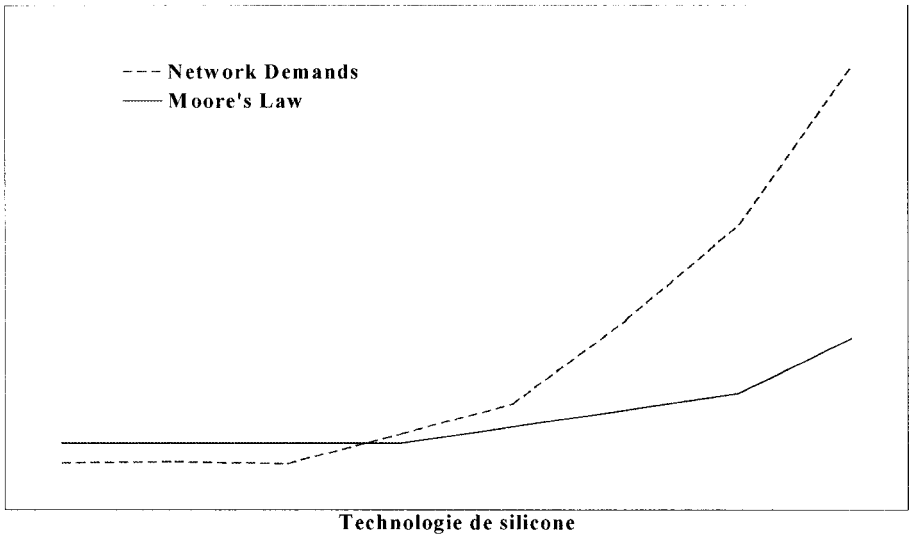


Figure 1.2 Vitesse des réseaux et demande de bande passante comparées à la progression des performances de la technologie d'intégration

1.1.2.2 Commutateurs réseau basés sur des circuits intégrés ASIC

Nous pouvons augmenter la vitesse d'un commutateur basé sur des processeurs RISC en ajoutant des accélérateurs matériels. Ces derniers peuvent copier des paquets à des vitesses pouvant dépasser 10 Gbps mais ils ne sont pas flexibles ni programmables.

Généralement, un circuit intégré ASIC est conçu pour exécuter des tâches très spécifiques, qui améliorent le niveau de performance du commutateur, avec une flexibilité pour diverses fonctions. Ces puces sont dédiées au support du trafic à très haute vitesse mais elles ne sont pas capables de traiter des tâches additionnelles de gestion de trafic, telles que la classification, les contrôles d'accès, ... Ce genre de tâches est désormais réservé au processeur central. Les fabricants utilisent ainsi des circuits intégrés pour différentes tâches avec un processeur RISC comme processeur central.

Bien que ces puces aient la capacité de fonctionner à des vitesses de l'ordre du Gigabit par seconde et du Petabit par seconde, elles sont très dispendieuses et leur temps de développement peut prendre entre 12 et 18 mois, ce qui allonge le temps de mise en marché du produit. De plus, une fois que les instructions et la logique sont figées dans le silicium, il est difficile d'ajouter de nouvelles fonctionnalités pour améliorer la performance. Ainsi, pour ajouter de nouvelles fonctions, le fabricant doit remplacer le circuit intégré, ce qui retarde de 6 à 9 mois la mise en marché du produit modifié.

1.1.2.3 Commutateurs réseau basés sur des processeurs RISC améliorés

Afin de résoudre les différentes lacunes qu'ont connu les commutateurs basés sur des processeurs RISC, les fabricants ont développé plusieurs techniques :

- ***Le partitionnement*** : Il s'agit de séparer diverses fonctions dans les équipements du réseau et de les dédier aux processeurs appropriés. Les fonctions de type transfert « forwarding », routage/signalisation « routing/signaling » ainsi que les applications de segmentation et de ré-assemblage de paquets dans les cellules IP/ATM (*Internet Protocol/Asynchronous Transfer Mode*) peuvent être séparées du processeur central.

- ***Le parallélisme*** : Il s'agit d'utiliser plusieurs processeurs pour améliorer la performance globale du système. Par exemple, dans les routeurs IP, nous séparons de plus en plus la fonction de transfert (forwarding) du routage en assignant chaque fonction à un processeur. Cette forme de parallélisme améliore nettement le réseau jusqu'au point où le taux de données d'une simple interface excède les capacités des entrées/sorties associées au processeur. Il est à noter que tout changement de fonctionnalité affecte le système, car, il faudrait rajouter des processeurs parallèles et les redistribuer.

- ***L'optimisation de la mémoire cache*** : Le défi avec les processeurs cache-centriques qui traitent des opérations intensives d'entrées/sorties repose sur l'optimisation. Cette dernière permet de réduire le nombre d'instructions en mémoire principale ainsi que celui dans la mémoire cache des processeurs. Ceci a pour effet de minimiser les erreurs qui résultent de temps de traitement excessif quand les données ou segments de programmes importants ne sont pas dans la cache.

- ***L'optimisation du bus*** : Il s'agit de la réduction de la quantité de données requises par le processeur sur le réseau. Ceci aura pour effet d'améliorer suffisamment la capacité des entrées/sorties associées au processeur.

- ***L'accélération par le biais d'un coprocesseur*** : Il s'agit de dédier les fonctions assez complexes à des coprocesseurs matériels. De nos jours, la segmentation et le rassemblage de paquets ATM sont traités par des accélérateurs matériels. Également, les fonctions de liaison de couches comme le HDLC (*High-level Data Link Control*) sont déléguées à des circuits intégrés dédiés.

1.1.2.4 Commutateurs réseau basés sur des approches hybrides

Les processeurs réseau pourraient être la solution à la demande grandissante de largeur de bande des applications réseau. Ils combinent la programmabilité, la flexibilité des processeurs RISC et la performance des circuits intégrés à l'intérieur d'un même produit. Ainsi, les tâches spécifiques sont attribuées aux circuits ASIC, tandis que le processeur RISC joue le rôle du processeur central. Un exemple de cette approche est le modèle ***CISCO EXPRESS FORWARDING*** [9] dont le processeur central détermine le chemin d'accès. Ensuite, il charge une copie complète de cette information dans la carte d'interface de ligne (ASIC) qui s'occupe de la commutation du trafic à très haute vitesse.

1.1.2.5 Architecture de processeurs intermédiaires

Cette architecture est basée sur le pipeline et sur la technique de séparation des bus pour les opérations d'entrées/sorties. Avec l'arrivée des processeurs de traitement numérique du signal DSP (*Digital Signal Processor*), nous constatons que le traitement des signaux s'améliore à mesure que le nombre d'entrées/sorties augmente. Par exemple, l'architecture Harvard [45] sépare le chemin d'instructions et de données de celui des bus d'entrées/sorties. Ceci a permis d'utiliser un bus séparé pour les opérations d'entrées/sorties qui pouvait être réduit en fonction de la bande passante des signaux. Il en découle l'élimination du comportement « non-déterministic » dans le cas où ce bus serait partagé avec le chemin d'instructions et de données. Par contre, les DSPs sont extrêmement difficiles à programmer avec des langages de haut niveau. Autrement dit, l'utilisateur doit avoir une bonne connaissance de l'architecture de son composant s'il veut tirer pleinement avantage de sa puissance.

Ainsi, le parallélisme et le pipelinage sont des techniques importantes dans la conception des architectures de processeurs réseau qui permettent d'améliorer la performance des modèles de la nouvelle génération.

1.2 Processeur réseau

Le processeur réseau se distingue de tout autre processeur par sa fonctionnalité versatile. Les fonctions d'un processeur optimisé pour les applications du réseau traitent des opérations qui favorisent une réutilisation de son système. Autrement dit, il est capable de s'adapter à des nouvelles fonctionnalités.

1.2.1 Définition d'un processeur réseau

Dans le temps des commutateurs et des ponts basés sur un logiciel informatique, le besoin d'un processeur réseau spécialisé ne se faisait pas sentir. Le processeur d'interface physique décodait les paquets d'entrées du réseau auquel il était connecté et il passait les données à traiter au processeur central. Les décisions de commutation du trafic étaient basées sur des instructions fournies par un logiciel assez complexe comme illustré dans la figure 1.1.

Afin de supporter des réseaux à très haut débit tels que, Fast Ethernet (100 Mbps) et Gigabit Ethernet (1 Gbps), des processeurs réseau spécialisés sont désormais requis pour obtenir une performance acceptable. Ces processeurs sont constitués de plusieurs circuits intégrés spécialisés dont la fonctionnalité repose en partie sur l'analyse, le traitement des paquets et la classification des protocoles réseaux à très haute vitesse.

Typiquement, les processeurs réseau se situent sur le chemin de données entre l'interface physique et la carte mère (« backplane »). Les principales fonctions exécutées par un processeur réseau sont les suivantes :

- ***Segmentation et rassemblage*** : les paquets sont désassemblés, analysés et ensuite, rassemblés pour être envoyés.

- ***Identification de protocoles et classification*** : en se basant sur les informations que les paquets contiennent, telles que le type de protocole, le numéro de port et l'adresse de destination, ils sont alors identifiés.

- ***Contrôle d'accès*** : après avoir été identifiés, les paquets sont placés dans des queues pour être manipulés, notamment, l'ajout de priorité ainsi que la gestion de trafic. Les paquets sont aussi vérifiés pour la sécurité informatique.

- ***Gestion du trafic*** : certains protocoles ou applications exigent une gestion du trafic pour assurer un délai ou une variation des délais convenables.

- ***Qualité du service*** : en plus de la gestion du trafic pour assurer une bonne qualité du service, les paquets devront être étiquetés pour un traitement plus rapide dans les dispositifs subséquents.

1.2.2 Caractéristiques d'un processeur réseau

Selon le point de vue des fabricants d'équipements réseaux, le processeur réseau est une composante clé qu'ils peuvent utiliser pour différencier leurs produits de leurs compétiteurs [2]. En livrant des produits intégrants des processeurs réseau avec des capacités avancées, les fabricants peuvent offrir des fonctions et une performance accrue qui peuvent valoriser leurs produits par rapport à leurs compétiteurs. Par conséquent, les vendeurs matériels sont à la recherche de processeurs réseau qui peuvent leur donner cet avantage. Ainsi, un processeur réseau requiert les caractéristiques suivantes :

- ***Programmabilité*** : le processeur réseau doit être facilement programmable pour tenir compte des nouvelles caractéristiques et de l'intégration rapide des technologies existantes et nouvelles. Il devrait rendre facile la conception de produits qui utilisent différents trafics.

- ***Performance*** : le processeur réseau doit s'adapter rapidement aux demandes de bande passante. Il doit être capable de supporter des milliers de connexions presque simultanées.

- ***Management*** : il s'agit de la gestion des tâches, telles que l'identification, la classification et le contrôle du trafic.

- ***Routage*** : les décisions de routage sont basées sur de l'information pré-programmée.

1.2.3 Le routage

Une étude de processeurs réseau ne peut être complète sans une analyse des détails pertinents du routage.

Le routage est une tâche parallèle, puisque chaque paquet qui traverse un réseau est indépendant avec son propre en-tête de destination et sa charge de données. Un routeur de petite capacité sur un ***LAN*** (*Local Area Network*) contrôle un nombre relativement petit de destinations, telles que des terminaux et des imprimantes locales. Par contre, un routeur de grande capacité sur un ***WAN*** (*Wide Area Network*) est responsable de la circulation de millions de paquets indépendants qui ont des milliers de possibilités de destinations, telles que les serveurs Internet et les serveurs de courriels. La commutation de paquets permet de traiter indépendamment plusieurs paquets en parallèle, sans tenir compte de l'ordre d'arrivée. L'appareil qui reçoit ces paquets, est responsable de les remonter dans leur ordre original. Pour acheminer un paquet vers sa destination, un routeur doit lire l'en-tête pour trouver l'adresse de destination, ainsi que d'autres informations. Ensuite, il le dirigera le plus rapidement vers la bonne destination. Il est à noter qu'un paquet peut comporter multiples en-têtes, notamment, un en-tête ***TCP***

(*Transmission Control Protocol*), un en-tête ***IP*** et un en-tête ***Ethernet*** dont la longueur variable.

De nos jours, les opérateurs de réseaux souhaitent une amélioration des routeurs en ce qui a trait à leur capacité de traitement des en-têtes, afin de prendre des décisions plus intelligentes dans le transfert des paquets, ce qui renforce la nécessité des processeurs réseau de haute performance.

1.3 Le marché des processeurs réseau OC-192

Le marché des commutateurs réseau est estimé à plus de 10 milliards de dollars US cette année, 50 % de plus que l'année passée. Les fabricants de processeurs réseau se préparent pour faire face à ce marché grandissant. Ils trouvent la nécessité d'introduire des produits complexes et bien différents des microprocesseurs standards comme moteurs pour les commutateurs de grande capacité.

Le marché des processeurs réseau a connu une forte percée technologique chez plusieurs compagnies bien établies comme IBM Corp., Agere Inc. (acquise par Lucent Technologies) et C-Port Corp. (acquise par Motorola Inc.). Une nouvelle vague de petites entreprises (start-up) visent aussi ce marché, telles que Bay Microsystems Inc., Lara Networks, Ezchip Technologies et Zettacom Inc.

L'explosion de l'Internet et du commerce électronique requiert le déploiement d'équipements d'une grande bande passante et d'une meilleure flexibilité afin de supporter les différentes technologies de l'Internet. Les capacités du processeur réseau peuvent aider à intégrer ces nouvelles technologies. Les fabricants de puces électroniques visent le marché du trafic OC-192 pour lequel les débits de données peuvent atteindre 10 Gbps. Les premiers processeurs réseau sont en production depuis la fin de l'année 2000. Nous présenterons dans les prochaines sections quelques-uns des processeurs réseau qui sont présentement sur le marché.

1.3.1 Alchemy (Au1000)

La puce de la compagnie Alchemy Semiconductor Inc. est surtout dédiée aux équipements d'accès [3]. Par contre, la compagnie prétend que son produit convient aussi pour les routeurs et les cartes d'interfaces. En fait, c'est une puce à faible puissance (300 mW) basée sur un processeur MIPS (*Millions of Instruction Per Second*) dont le jeu d'instructions est augmenté de quelques nouvelles instructions. Également, plusieurs variétés d'interfaces sont intégrées.

1.3.1.1 Architecture interne

Le « Au1000 » est basé sur une machine MIPS de 32-bit. Ce processeur a une architecture pipeline de cinq étages, optimisée pour minimiser les pénalités de branchement. L'architecture contient aussi un multiplieur-accumulateur de 32-bit qui fonctionne en parallèle avec le CPU pipeliné. De plus, cette puce supporte des instructions spéciales pour le déplacement conditionnel entre les registres et la pré-recherche de l'instruction dans la mémoire.

L'architecture contient aussi deux contrôleurs Ethernet, un port infrarouge, un port USB et quatre unités asynchrones de transmission et de réception UART (*Universal Asynchronous Receiver and Transmitter*). Il existe aussi une mémoire cache d'instructions et de données de 16Ko. L'architecture interne du Au1000 est montrée à la figure 1.3.

1.3.1.2 Programmabilité

Étant donné que cette puce est basée sur une machine MIPS, nous pouvons la programmer en langage C. Un outil de développement logiciel est aussi fourni pour

programmer les applications. Les différents systèmes d'exploitation sont supportés tels que MS Windows CE, Linux et VxWorks.

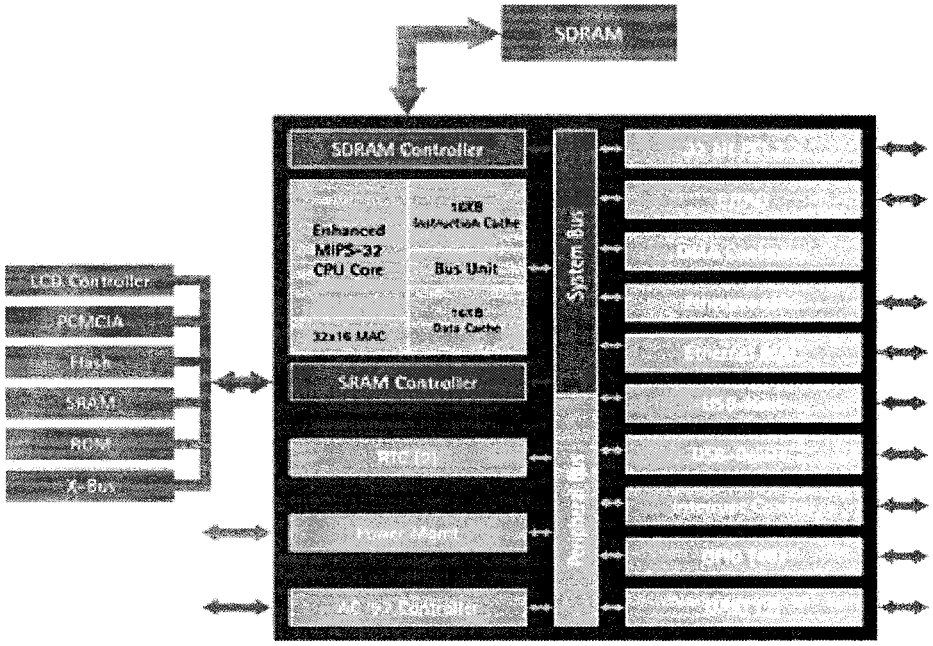


Figure 1.3 L'architecture interne du Au1000 [3]

1.3.1.3 Implantation

Le Au1000 est disponible comme un « soft-core » supportant plusieurs fréquences d'opération, notamment 266 MHz, 400 MHz et 500 MHz. La puissance totale dissipée est alors inférieure à 300 mW, 500 mW et 900 mW respectivement.

1.3.2 IBM (PowerNP)

La puce électronique fournie par IBM est une solution caractérisée par un processeur central PowerPC, 16 processeurs de protocoles qui sont inclus à l'intérieur d'un processeur embarqué complexe et 7 coprocesseurs matériels [20]. Elle supporte la technologie « Packet over SONET (*Synchronous Optical Network*)» et GigaBit Ethernet

avec une vitesse pouvant atteindre 2.5 Gbps. De plus, cette puce est ciblée pour le traitement des paquets entre les couches 2 à 5 du modèle OSI.

1.3.2.1 Architecture interne

L'architecture interne de ce processeur réseau est constituée d'un processeur embarqué complexe, d'une partie matérielle spéciale et de plusieurs interfaces. Le cœur du processeur réseau est un processeur PowerPC405 [21] avec 16 processeurs de protocoles programmables. Chaque paire de processeurs de protocoles partage un coprocesseur matériel pour la recherche d'arbre et la modification des trames. Un processeur de protocoles possède un pipeline à trois étages. Il existe aussi sept coprocesseurs spécialisés dédiés à chacun des 16 processeurs de protocoles, dont les fonctions sont les suivantes : le rangement de données, le calcul du « checksum », le contrôle de flot, la mise à jour des processeurs de protocoles, les interfaces et la manipulation des données. Chaque processeur de protocoles possède une mémoire cache d'instructions de 8 Ko. La figure 1.4 illustre l'architecture interne d'un IBM-PowerNP.

1.3.2.2 Programmabilité

L'outil de développement logiciel inclut un assembleur, un dévermineur et un simulateur.

1.3.2.3 Implantation

La puce est implantée en utilisant la technologie CMOS 0.18µm. Elle dissipe une puissance de 20W et opère à une fréquence de 133 MHz [15].

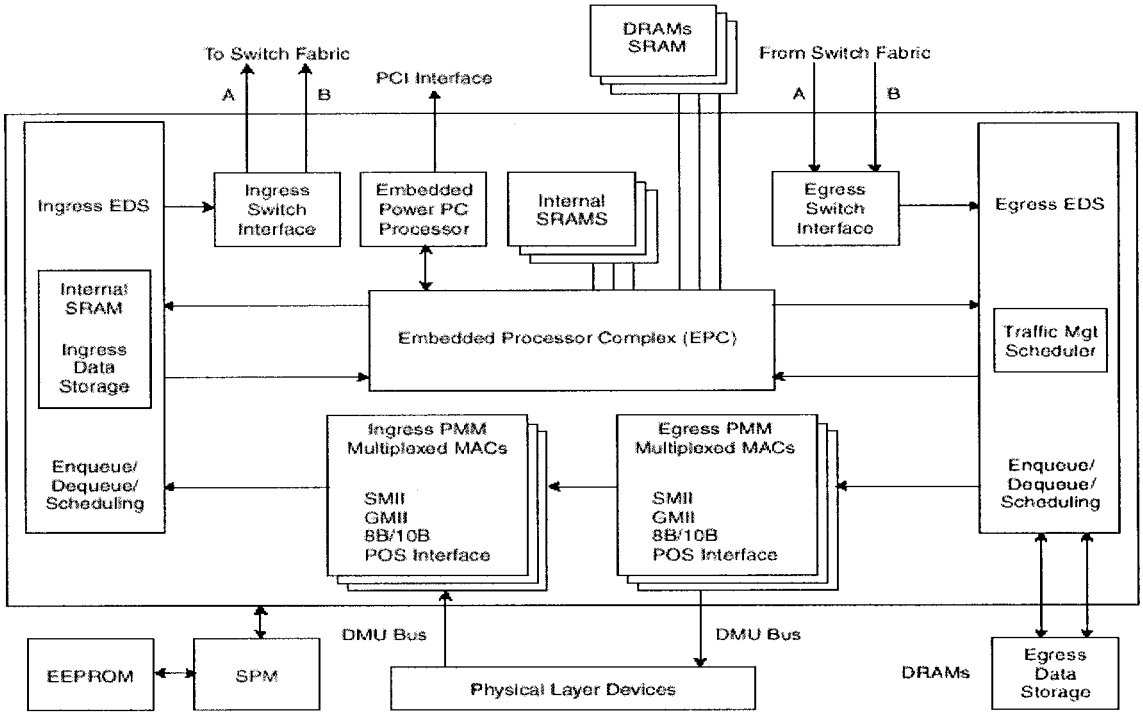


Figure 1.4 Schéma bloc de l'architecture interne d'un processeur réseau IBM-PowerNP [20]

1.3.3 Intel (IXP1200)

La société Intel est un des pionniers dans le domaine des processeurs réseau. Le IXP1200 cible la manipulation des paquets des couches 2 à 4 du système OSI. Il supporte un débit de 2.5 Mpaquets/s. Les couches supérieures peuvent être supportées en connectant des processeurs externes par l'intermédiaire de l'interface PCI (*Peripheral Connection Interface*). Le IXP1200 consiste en six « micro-engins » et un processeur central StrongARM qui agit comme contrôleur. Les « micro-engins » ont un support matériel pour servir jusqu'à quatre « threads ». De plus, il contient des coprocesseurs matériels pour calculer des fonctions spéciales telles que le décalage en un cycle, le « queuing » et le « hash ».

1.3.3.1 Architecture interne

L'architecture est composée de six processeurs embarqués programmables et d'un processeur central StongARM qui fonctionne à 200 MHz et qui joue le rôle d'administrateur de système [24]. Un bus de 64-bit fournit une large bande passante nécessaire pour assurer une bonne connectivité entre les six processeurs embarqués, le processeur central, la mémoire et tous dispositifs externes tels que les interfaces physiques. D'autre part, le bus PCI permet une intégration avec un processeur externe de contrôle.

Les six processeurs embarqués exécutent toutes les tâches de manipulation sur les paquets. En fait, ils ont un support matériel pour effectuer quatre tâches chacun. Ceci nous offre un grand total de 24 tâches exécutables simultanées. Bien que les quatre tâches qui roulent sur un même processeur embarqué partagent le même fichier de registres temporaires, le logiciel divise ce dernier en quatre sections pour chaque tâche. Ceci est possible grâce à la capacité des micro-engins d'exécuter un changement de contexte en un seul cycle.

Le IXP1200 contient aussi des coprocesseurs matériels pour accélérer certaines fonctions. Il faut noter l'existence d'une machine programmable pour effectuer la fonction de « hash » ainsi que des queues qui sont partagées par les six processeurs embarqués et le processeur central. De plus, nous y trouvons des mémoires de type FIFO (*First In First Out*) qui agissent comme interfaces aux dispositifs d'interfaces physiques dans les cas de lecture et d'écriture. Finalement, nous dénotons l'existence d'une mémoire cache de 8 Ko pour les données et une autre mémoire cache de 16 Ko pour les instructions. La figure 1.5 illustre le schéma bloc de cette architecture.

1.3.3.2 Programmabilité

La programmation du IXP1200 se fait en langages assembleur et C [12]. Étant donné que les six processeurs embarqués travaillent en même temps, la tâche de programmation est très ardue. Il est aussi utile de mentionner que, malgré la difficulté rencontrée lors de la programmation, l'outil de développement fournit une aide extraordinaire pour faciliter à l'utilisateur la tâche de programmation. L'environnement de simulation configurable et la visualisation claire, montrant toutes les activités sur la puce, facilitent le déverminage.

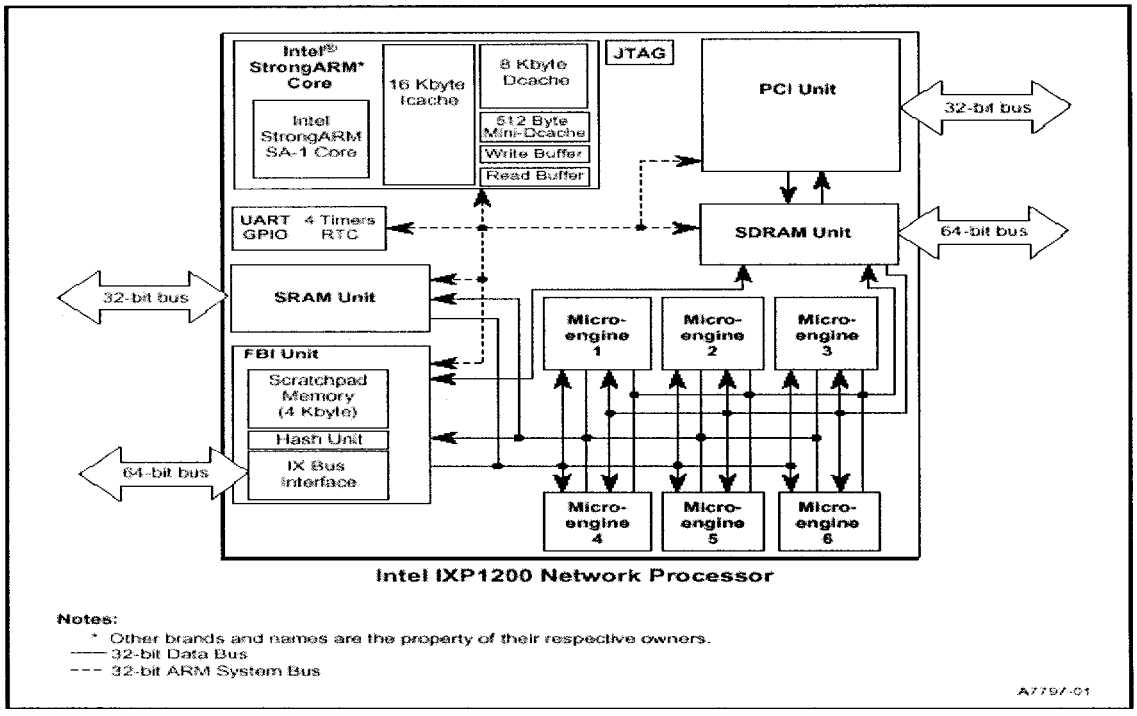


Figure 1.5 Schéma bloc de l'architecture interne du Intel-IXP1200 [24]

1.3.3.3 Implantation

Le IXP1200 a été implanté en utilisant la technologie 0.18 µm. Il opère à une fréquence de 200 MHz tout en dissipant une puissance de 5 W [15].

1.3.4 Motorola (C-5 DCP)

Le processeur réseau C-5 de la compagnie Motorola est constitué de 16 processeurs avec cinq coprocesseurs et d'un processeur central qui agit comme coordonnateur de système [19] et [29]. Chaque processeur est un RISC avec deux processeurs de données série. Ce processeur réseau cible la manipulation de paquets des couches 2 à 7 du modèle OSI avec un taux de 2.5 Gbps.

1.3.4.1 Architecture interne

Les processeurs embarqués sont dédiés à la caractérisation et la classification des paquets ainsi que la gestion du trafic. Par contre, les deux processeurs de données série sont dédiés à la vérification des en-têtes, à l'extraction, à l'insertion, au calcul et à la vérification du CRC (*Cyclic Redundancy Check*). Les cinq coprocesseurs ont chacun des fonctions différentes, notamment, la mise à jour des tables de correspondance, la gestion des queues des paquets, la gestion de la mémoire, la coordination avec d'autres processeurs externes et l'utilisation de plusieurs processeurs réseau.

De plus, il existe trois bus internes qui ont une bande passante de 60 Gbps. L'architecture interne est montrée à la figure 1.6.

1.3.4.2 Programmabilité

Le C-5 DCP est programmable en utilisant le langage C/C++. L'outil de développement contient un ensemble vaste de plusieurs applications communes [30].

1.3.4.3 Implantation

Ce processeur réseau est implanté avec la technologie 0.18 µm et dissipe une puissance typique de 15 W.

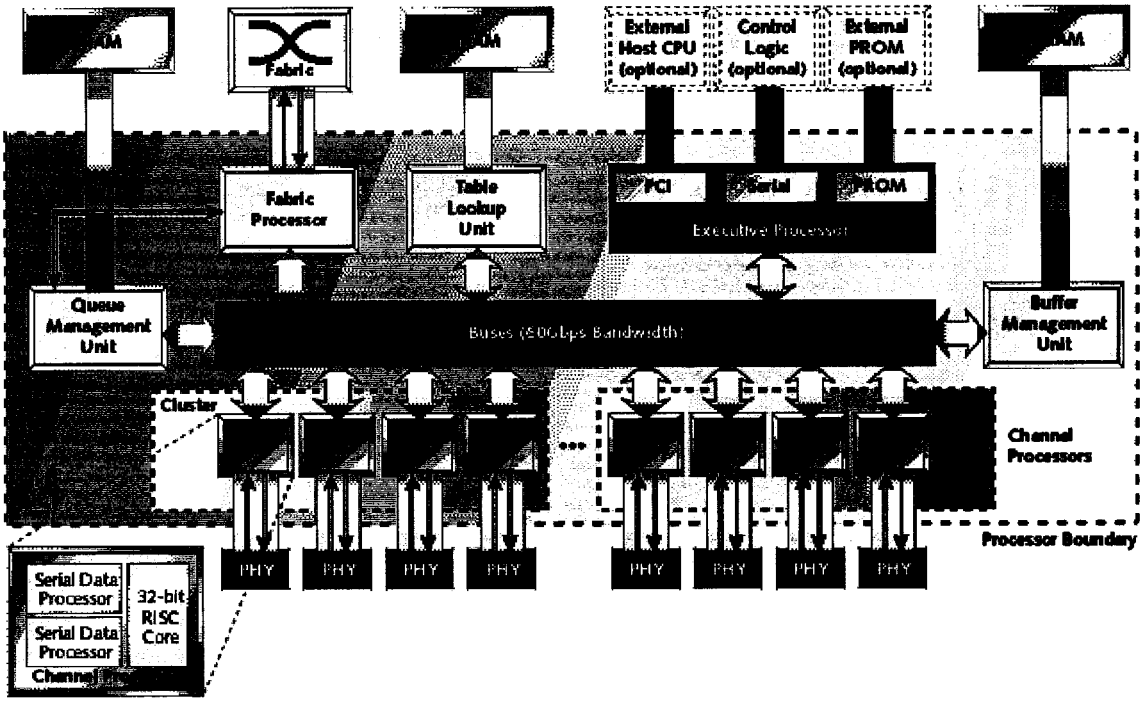


Figure 1.6 Schéma bloc de l'architecture interne du processeur réseau de Motorola [19]

1.3.5 Autres processeurs réseau

Nous avons aussi fait l'étude de plusieurs autres processeurs réseau dont voici une liste avec les pointeurs respectifs :

- **Agere (PayloadPlus)** [27];
- **AMCC (nP7xxx)** [31];
- **Bay Microsystems** [46];
- **BRECIS Communications (MSP5000)** [8];

- **Broadcom (Mercurian SB-1250)** [37] et [38];
- **ClearSpeed** [11];
- **ClearWater Networks (CNP810SP)** [36]
- **Cognigine** [35];
- **Conexant (MXT4400)** [28];
- **Ezchip (NP-1)** [13] et [14];
- **Lexra (NetVortex, NVP)** [4] et [16];
- **PMC-Sierra** [33];
- **Vitesse (PRISM IQ2000)** [39];
- **Xelerated Packet Devices (T40, X40)** [47] et [48].

1.3.6 Synthèse de trois principaux processeurs réseau

Les processeurs réseau possèdent des éléments de conception communs :

- Accélération matérielle des tâches effectuées sur la majorité des informations traitées (ex. : CRC, queue management);
- Extensibilité (*Scalabality*) : rend possible l'augmentation de la puissance de traitement par la collaboration de plusieurs exemplaires d'un élément;
- La majorité des processeurs contiennent un processeur générique pour le traitement des tâches de haut-niveau.
- System-on-a-chip (SOC) : combinaison de processeurs, mémoire et interface d'entrée/sortie sur une même pièce de silicone. Parmi des avantages associés à cette technologie : délai de propagation réduit entre les composantes et horloge plus rapide.

Le tableau suivant résume les caractéristiques de trois processeurs réseau principaux.

Tableau 1.1 Synthèse et comparaison des trois principaux processeurs réseau

	Intel IXP1200	IBM NP4GS3	Alchemy AU1000
Processeur générique	Intel StrongARM	Power PC 405	MIPS 32
Coprocesseurs spécialisés	Hash Unit Queue Management	Tree searching Checksum Enqueue Interface String Copy Counter Policy *Dédié à chaque picoprocesseur*	Aucun
Unités fonctionnelles programmables	Microengines (6x)	Pico-processeurs de protocoles (16x)	Aucune
Vitesse de traitement	6.6 Gbps peak	4.5 Mpps	N/D
Bus interne	ARM AMBA	N/D	N/D
Interfaces	POS MAC PCI UART GPIO	POS MAC PCI DASL	MAC IrDA SSI UART GPIO USB
Applications	Multi-layer LAN switches Multi-protocol telecommunications products Broadband cable products Remote access devices Intelligent PCI adapters	Layer 2 and Layer 3 switching Packet classification Multiple table lookups per frame Packet Modification Queue/Policy management General packet processing	High performance at low power PDA's Wireless / remote access Routers Line cards
Protocoles supportés	10/100 Mbps Ethernet Gigabit Ethernet UTOPIA/POS	10/100 Mbps Ethernet Gigabit Ethernet OC-12 POS OC-3 POS 802.3ad (Link Aggregation) 802.1q (VLAN)	10/100 Mbps Ethernet USB

Dans ce chapitre, nous avons fait un survol des différentes architectures de commutateurs. Afin d'assurer une flexibilité et une bonne programmabilité d'un processeur général tout en maintenant la performance à une vitesse atteignant 1 Gbps, il faudrait regarder au-delà des architectures basées sur des processeurs RISC . De nouvelles machines telles que les processeurs réseau programmables offrent une meilleure solution. Ces machines doivent combiner, d'une façon innovatrice, le pipeline, l'exécution parallèle et le déploiement matériel des nouveaux algorithmes. Une architecture réussie de processeur réseau doit tenir compte de la performance exigée de nos jours et de la flexibilité de rajouter des fonctionnalités futures.

CHAPITRE II

Architecture globale de la plate-forme SoC

L'objectif de ce chapitre est d'exposer l'architecture flexible d'une plate-forme SoC, spécialisée dans le transfert de données vidéo de qualité studio. Cette application nous guide vers un compromis entre la flexibilité et la performance [2]. Des travaux préliminaires nous ont conduits à proposer cette architecture. Comme les gains en performance sont, au mieux, linéaires avec le nombre de processeurs [18], nous avons considéré une architecture basée sur des coprocesseurs. Avec cette approche, il est nécessaire de savoir quelles fonctions se présentent comme des goulots d'étranglement dans le processus de conversion. Un coprocesseur est typiquement introduit pour résoudre à chacune de ces difficultés.

Le modèle exécutable de l'architecture proposée, sera mené en proposant une description fonctionnelle des modules qui simuleront la conversion de paquets [40]. Les spécifications devront définir une architecture permettant la conversion unidirectionnelle de quelques protocoles.

2.1 Architecture interne

2.1.1 Première version du modèle exécutable et processus de conversion

Les étapes pour la définition des spécifications de l'architecture ont été nombreuses. La première version, proposée à la figure 2.1, n'est pas la plus complexe et tient compte d'un certain nombre d'hypothèses simplificatrices. Toutefois, elle est le résultat d'évolutions,

avec la modification, le retrait ou l'ajout de modules, par rapport à une proposition plus générale présentée en figure 2.2.

La figure 2.2 présente l'architecture la plus aboutie. Par rapport à la première version, les modules comme le processeur ***ARM***, le « ***Core Connect Bus*** » d'***IBM*** ou les interfaces d'entrées et de sorties devront être pris en charge dans le cadre de travaux futurs. En effet, ceux-ci ne constituent pas des priorités pour la réalisation du modèle exécutable. Cette architecture complexe a été établie de sorte à prendre en considération les évolutions possibles de la plate-forme SoC.

La différence essentielle entre les 2 figures réside dans le fait que nous nous proposons d'utiliser, dans une première approche, un seul engin de formatage ***GF*** (*General Formatter*). Par conséquent, le bus d'***IBM*** perd son intérêt, compte tenu du fait qu'il nécessite un important apprentissage additionnel. Les interfaces seront considérées comme des IPs qui devraient être disponibles sous peu dans les bibliothèques de founisseurs d'outils comme ***Synopsys***. Le processeur ***ARM*** requiert aussi une étude très approfondie.

Le processus de conversion relatif à notre architecture est le suivant :

- Le gestionnaire de mémoire stocke les données en mémoire principale et envoie en même temps au contrôleur des informations concernant les données mémorisées.

- Parallèlement, le contrôleur met à jour des étiquettes relatives au paquet dans le registre d'index.

- À partir d'un certain nombre de données stockées, nous sommes en mesure de déterminer quels protocoles sont présents dans chacune des couches du paquet qui nous intéresse.
- Dès lors, le contrôleur prévient le GF qu'il peut commencer son traitement de conversion. La conversion ne se fait que sur les en-têtes des couches concernées. La charge utile demeure dans la mémoire principale.

- Le GF exécute le programme de conversion et est aidé par le coprocesseur de vérification/génération d'adresses. Les en-têtes convertis sont stockés dans la mémoire temporaire de sortie.

- Lorsque la conversion des données est terminée, le GF prévient le coprocesseur d'assemblage afin qu'il effectue l'assemblage des données.

- Le tout est envoyé au coprocesseur d'intégrité qui, en calculant le « checksum » et le CRC, ajoute les queues pour chacune des couches. L'interface de sortie fournit ensuite la donnée convertie au réseau.

Des détails sur chacun des modules seront donnés dans la partie 2.1.3.

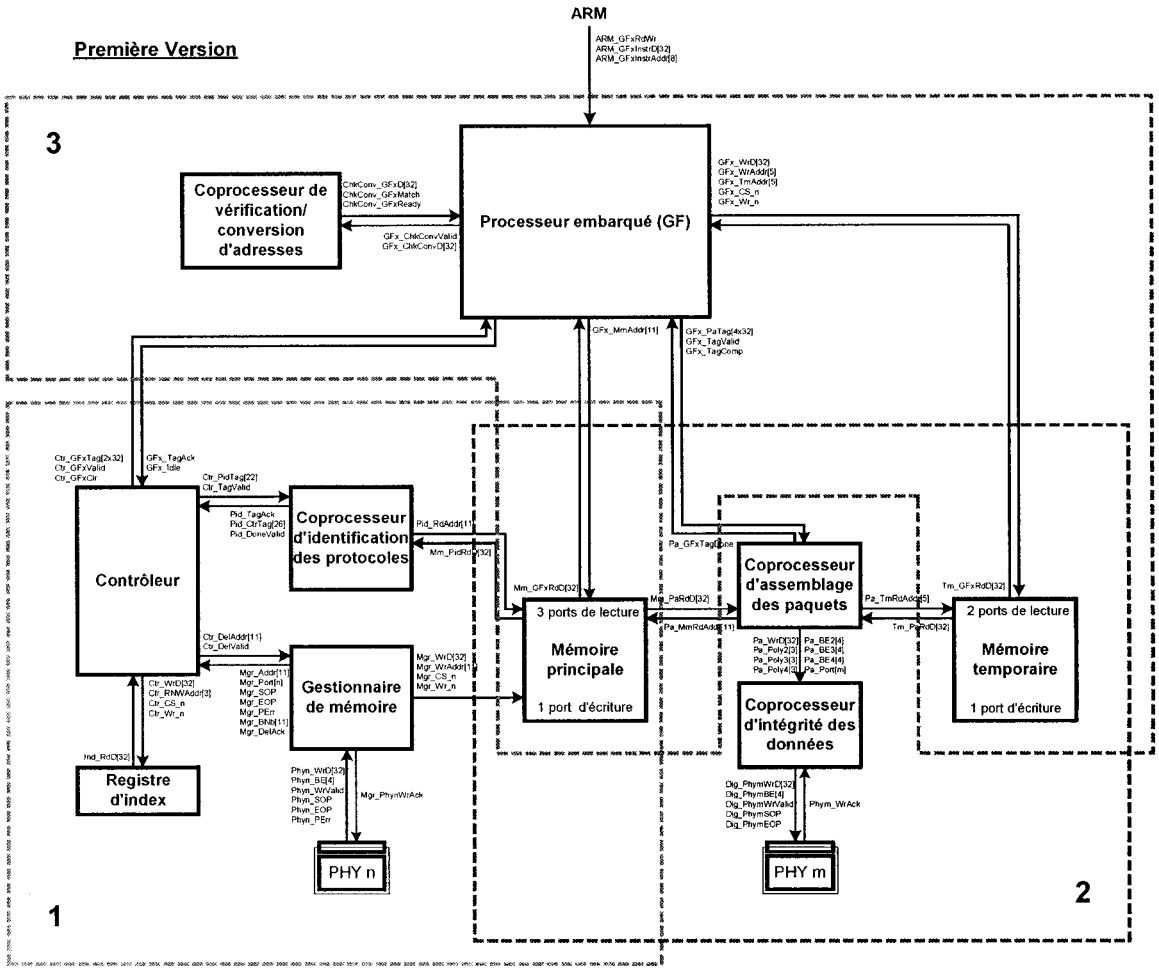


Figure 2.1 Architecture de la première version

Les blocs délimités en pointillé correspondent aux parties de l'architecture qui seront intégrées et testées indépendamment. Nous notons, en effet, que ces trois parties communiquent entre elles, soit par la mémoire multiport principale, soit par l'intermédiaire d'un nombre restreint de signaux. Les signaux provenant du processeur ***ARM*** seront, en fait, envoyés par l'utilisateur pour la première version.

2.1.2 Évolution de l'architecture

Version simplifiée

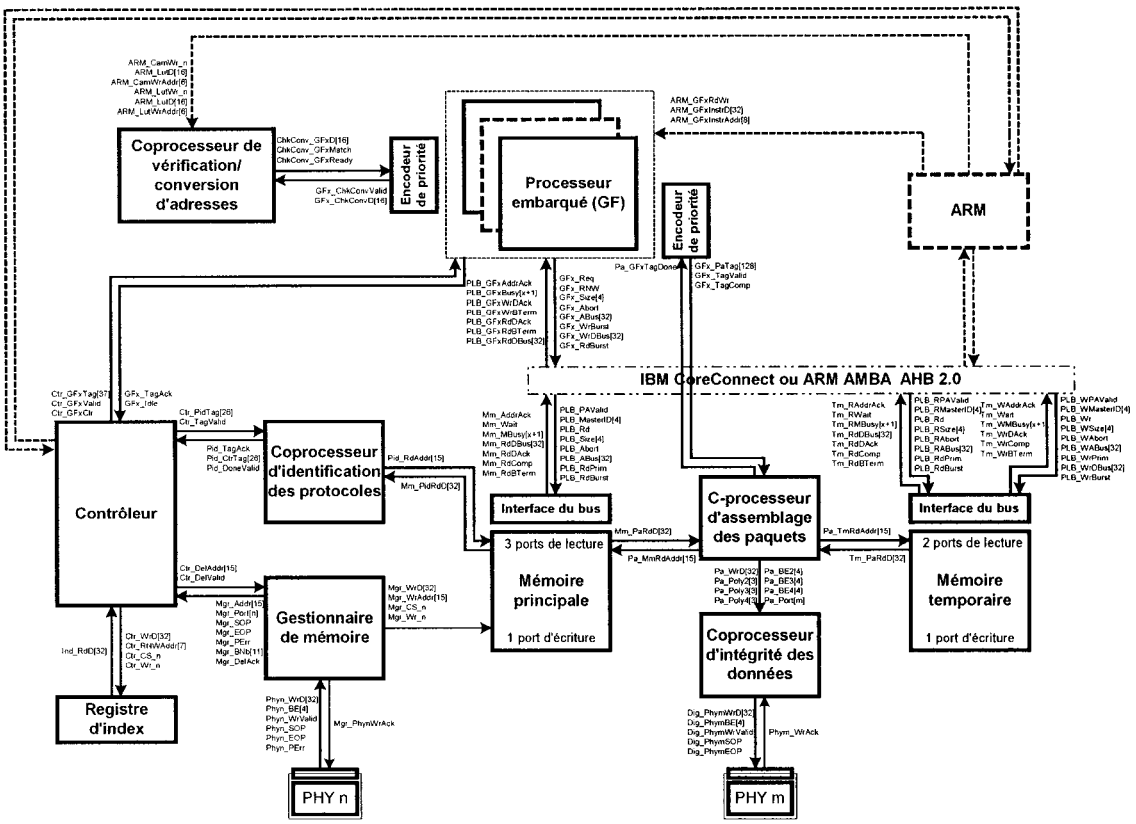


Figure 2.2 Architecture globale

2.1.3 Description des rôles des différents modules

2.1.3.1 Les interfaces

De façon générale, les interfaces permettent la réception et la transmission de données avec le monde extérieur. Chaque interface doit être mise en œuvre selon les règles de l'art des spécifications des signaux du médium physique (PHY) et des services de la couche de liaison des données DLL (*Data Link Layer*) définies par les standards correspondants.

Les interfaces sont en mesure de :

- Transformer les signaux analogiques en signaux logiques et vice versa;
- Générer des fanions identifiant le début et la fin d'un paquet ainsi que d'autres renseignements utiles;
- Récupérer l'horloge incluse dans les signaux reçus;
- Encoder les signaux logiques à transmettre de façon à ce que leur fréquence de transmission (horloge) soit récupérable;
- Rendre transparente la gestion des services offerts par les couches physique et de liaison de données.

Les interfaces sont des modules très complexes, faisant intervenir de l'électronique analogique et numérique. Dans le cadre de ce projet, aucune interface ne sera réalisée. Cependant, une recherche s'est révélée nécessaire pour savoir la forme sous laquelle se présenteront les données qui subiront un traitement par le processeur réseau. Leur étude nous permet de réaliser qu'elles sont très différentes les unes des autres. En effet, il n'y a pas une interface qui possède les mêmes signaux de contrôle pour communiquer avec le circuit hôte. Certaines interfaces intègrent des FIFO pour synchroniser la lecture des données avec l'horloge du circuit hôte. D'autres exigent de l'hôte de fournir ce circuit.

Seul point important en commun, toutes les interfaces étudiées possèdent un mécanisme de vérification et de génération de l'intégrité des données. Ainsi, lors de la réception d'un paquet, les interfaces vérifient les données et génèrent un signal indiquant sa validité. À la transmission, elles s'occupent de générer et de placer la séquence de vérification des données à la fin des paquets.

Dans le cadre de notre projet, ces signaux d'interface seront simulés par notre banc d'essai. Nous admettons aussi que ces interfaces sont capables de vérifier ou générer des CRCs, respectivement en entrée ou en sortie. Des modèles comportementaux d'interfaces physiques ***IEEE 802.3*** et ***IEEE 1394*** sont déjà rendus disponibles par la ***CMC***

(*Canadian Microelectronics Corporation*) et que nous pourrons l'intégrer dans notre plate-forme avec ***Synopsys*** ou ***Seamless***.

2.1.3.2 Le gestionnaire de mémoire

Le gestionnaire reçoit les données des paquets transmis par l'interface physique d'entrée. Il se charge de stocker ces données dans la mémoire principale. Parallèlement, il envoie un certain nombre d'informations au contrôleur (nombre de données stockées, début et fin des paquets, etc.).

2.1.3.3 Les mémoires

Les mémoires sont au nombre de trois. Il s'agit de la mémoire principale, la mémoire temporaire et le registre d'index. Ce sont toutes des mémoires synchrones. Les deux premières sont multiport, c'est-à-dire qu'elles possèdent plusieurs ports de lecture. Quant au registre d'index, il stocke les informations envoyées (étiquettes) par le contrôleur concernant les paquets en mémoire principale.

2.1.3.4 Le contrôleur

Le contrôleur du convertisseur de protocoles gère les traitements sur les nouveaux paquets, depuis son écriture dans la mémoire jusqu'à la fin de son traitement par l'engin de formatage. Ses tâches spécifiques sont :

- Gérer la création d'étiquettes pour les nouveaux paquets;
- Amorcer l'identification du protocole du paquet dès qu'il y a un nombre suffisant d'octets en mémoire;
- Effectuer les mises à jour dans l'étiquette pour la taille finale du paquet;
- Envoyer au GF les informations sur le paquet et lui donner le signal de départ pour le traitement du paquet;

- Gérer la destruction en mémoire du paquet à la fin du traitement ou en cas de non-intégrité;
- Créer un lien entre le registre d'index et les autres modules.

2.1.3.5 Le coprocesseur d'identification des protocoles

Ce coprocesseur permet d'identifier les protocoles des trois couches (la couche liaison de données, la couche réseau et la couche transport) OSI voir 2.2.2.1) que supporte la plate-forme.

Il permet d'identifier les types de protocoles encapsulés dans un paquet à convertir. Cette identification est nécessaire pour le début de l'exécution du programme de conversion par l'Engin de Formatage (GF).

2.1.3.6 Le coprocesseur de vérification / conversion d'adresses

Le coprocesseur de vérification et de conversion des adresses assiste les engins de formatage dans le traitement des paquets. Ce module est subdivisé en deux sous-modules, le module de vérification d'adresses et le module d'identification de protocoles. Les adresses en question sont les adresses IP (source et destination).

2.1.3.7 Le coprocesseur d'assemblage des paquets

Ce coprocesseur a comme fonction d'aller chercher les en-têtes, traités par le ***GF***, dans la mémoire temporaire et la charge utile laissée dans la mémoire d'entrée principale, selon les informations données par le ***GF*** responsable du paquet à assembler. De plus, il assemble ces données et les transfère au coprocesseur d'intégrité des données dans l'ordre suivant : en-tête DLL, en-tête NL (*Network Layer*), en-tête TL (*Transport Layer*) et charge utile.

Cette tâche est effectuée par une machine à états qui va réaliser de façon cyclique les opérations mentionnées ci-dessus. En effet, il est plus avantageux d'utiliser une machine à états qu'un processeur complexe comme un ***GF*** ou un ***ARM***, car les opérations à effectuer sont répétitives.

2.1.3.8 Le coprocesseur d'intégrité des données

Ce coprocesseur transmet vers l'interface physique de sortie toutes les données d'un paquet converti et assemblé par le coprocesseur d'assemblage. Il ajoute principalement les queues de chacune des couches du paquet. Celles-ci sont souvent réduites à la génération de CRC 32, qui permettraient à l'équipement destinataire, la vérification de l'intégrité des données renvoyées au réseau.

2.1.3.9 Le « Core Connect Bus » d'IBM

Dans le cas où plusieurs engins de formatage et le processeur ARM seraient utilisés, un bus doit être utilisé pour leur permettre d'avoir accès aux mémoires. Le « Core Connect Bus » d'IBM permet de faire communiquer 8 maîtres, de façon transparente pour chacun, à une fréquence de 200MHz. Présentement, il existe deux bus de communication populaires commercialement disponibles pour les architectures embarquées. Il y a le bus ARM AMBA 2.0 et le IBM CoreConnect.

Le tableau suivant résume les principales caractéristiques et ce qui différencie ces deux bus :

Tableau 2.1 Comparaison entre le bus ARM AMBA 2.0 et le bus IBM CoreConnect

	ARM AMBA 2.0	IBM CoreConnect
Largeurs possibles du bus	32, 64, 128-1024	32, 64, 128, 256 bits
Bus de données	Lecture et écriture séparées	Lecture et écriture séparées
Particularités	Plusieurs maîtres (max. 16) Pipeline Transactions séparées Transferts en rafale Transferts de ligne	Plusieurs maîtres (max. 16) Pipeline à 4 niveaux (lecture) Pipeline à 2 niveaux (écriture) Transactions séparées Transferts en rafale Transferts de ligne
	Bus périphérique sur la puce	Bus périphérique avancé
Nombre de maîtres	Un seul, APB	Plusieurs
Fonction du pont	APB est le seul maître	Maîtres sur les deux bus
Bus de données	Séparés ou 3 états	Lecture et écriture séparées

En résumé, le bus que propose IBM est plus flexible que celui de ARM. En effet, il permet un plus grand nombre de maîtres et un grand nombre d'esclaves. Cette technologie est disponible à l'École Polytechnique de Montréal.

2.1.3.10 Le processeur ARM

Ce module contient un microprocesseur ARM7TDMI. Ce dernier s'occupe du traitement des tâches exceptionnelles. Les tâches énumérées jusqu'à présent sont :

- Configuration et initialisation des modules du système (notamment l'écriture du programme de conversion dans la mémoire d'instructions de l'Engin de Formatage);
- Génération de paquets destinés à l'interface de sortie;
- Réparation des paquets erronés mais corrigeables;
- Interface usager : affichage d'erreurs, activation/désactivation de certaines fonctionnalités.

2.2 Caractéristiques générales de la première version

Les paramètres et buts fixés pour la réalisation de la première version du modèle exécutable de notre convertisseur de protocoles sont les suivants :

- Une grande flexibilité (exécution d'un programme de conversion par l'engin de formatage);
- Une architecture entièrement paramétrable de sorte à obtenir un résultat générique et réutilisable (utilisation d'un « package »);
- Une latence la plus réduite possible, c'est-à-dire que le temps entre l'entrée du paquet à convertir et la sortie du paquet converti doit être minimal;
- Conversion sur les couches lien de données et transport;
- Conversion de paquets et non de couches du modèle OSI;
- Transfert de données vidéo numérique de qualité studio;
- Fréquence d'opération du système de 200MHz, voir la démonstration à la section 2.2.1.3.

2.2.1 Évaluation des communications

Un des objectifs de ce projet est la transmission temps-réel de vidéo numérique de qualité studio sur des réseaux locaux. Il est donc d'intérêt de choisir des protocoles de communication à haut débit capables de transporter ce genre d'informations. Nous souhaitons donc que les réseaux soient en mesure de transporter des charges d'un débit d'au moins 360 Mbps. Dans cette optique, nous devons nous assurer que les protocoles choisis sont en mesure de respecter cette contrainte.

2.2.1.1 Évaluation de la largeur de bande utile disponible d'un réseau : analyse et comparaisons

L'évaluation de la largeur de bande utile disponible d'un réseau nous permet de s'assurer qu'il sera possible de transmettre la quantité d'informations que nous désirons. Pour ce faire, il est nécessaire d'évaluer, dans un premier temps, le taux d'arrivée des paquets. Nous émettons l'hypothèse simplificatrice que la transmission de vidéo s'effectuera toujours dans un paquet de taille maximale. La validité de cette hypothèse sera révisée plus tard. Aussi, nous considérons que les réseaux opèrent à un régime équivalent à 80 % de leur bande passante maximale ou moins. De cette façon, la probabilité qu'un paquet transmis soit détruit par des équipements réseaux intermédiaires reste faible et sera négligée. La formule suivante permet de trouver le taux d'arrivée des paquets, exprimée comme une fonction de la largeur de bande (LB) utile d'un type de réseau :

$$\text{Nombre de paquets/sec} = \frac{80\,\% \times \text{LB du réseau}}{\text{Taille des paquets}}$$ où LB est la largeur de bande.

Dans un deuxième temps, nous devons évaluer la charge utile que contient un paquet de tailles maximales.

$$\text{Charge utile/paquet} = \begin{matrix}\text{Taille} \\ \text{d'un paquet}\end{matrix} - \begin{matrix}\text{Entêtes des couches} \\ \text{TL, NL et DLL}\end{matrix}$$

Finalement, nous pouvons évaluer la largeur de bande utile disponible en multipliant le taux d'arrivée des paquets avec la charge utile d'un paquet.

$$\text{LB utilisée} = \frac{\text{Nombre de paquets}}{\text{seconde}} \times \frac{\text{Charge utile}}{\text{paquet}}$$

En considérant que l'en-tête des paquets pour les protocoles des couches réseau et transport sont respectivement 160 bits pour IP et 64/96 bits pour UDP/RTP (*User Datagram Protocol/Real-Time Transport Protocol*), le tableau 2.2 présente le résultat des calculs qui donnent la bande passante utilisée pour les protocoles choisis.

Tableau 2.2 Largeurs de bande utilisées pour des paquets de tailles maximales

Protocole	Largeur de bande (Mbps)	Taille des en-têtes (bits)			Taille maximale d'un paquet (bits)	Nombre de paquets/sec	Charge utile (bits/paquet)	LB utile disponible (Mbps)
		DLL	NL	TL				
IEEE 802.3	1000	144	160	96	12144	65876	11680	769
IEEE 1394b	800	192	160	96	32960	19417	32448	630
IEEE 1394a	400	192	160	96	16576	19305	16064	310

Les résultats du calcul de la largeur de bande utile disponible montre bien, que même si le protocole ***IEEE 1394a*** possède une largeur de bande de 400 Mbps, une fois que la charge des en-têtes est enlevée à la taille du paquet, il n'est plus possible de transmettre de la vidéo numérique de qualité studio qui requiert 360 Mbps, tout en respectant un taux d'utilisation réseau de 80 %.

La raison pour laquelle nous avons limité le débit maximal à 360 Mbps est due au fait nous nous sommes attardés aux applications de type vidéo numérique, par exemple, HDTV (High Definition TV) compressée ou non-compressée. Cette dernière nécessite une large bande passante de l'ordre de 1 Gbps à 15 Gbps. En fait, l'ajout de l'interface sérielle numérique SDI (Serial Digital Interface) [44] nécessaire à la réception des images par un équipement HDTV, est considéré comme le goulot d'étranglement. La limite de cette interface est de 360 Mbps. Ceci explique la limite de 360 Mbps dans le débit maximal.

2.2.1.2 Évaluation de la largeur de bande nécessaire pour transmettre de la vidéo numérique de 360 Mbps

Maintenant que nous connaissons la largeur de bande utile disponible des réseaux, il n'est plus nécessaire d'utiliser des paquets de tailles maximales. Ainsi, il sera possible de poursuivre l'étude de la largeur de bande des canaux de communication du processeur réseau à l'aide de paquets de tailles minimales. Notons que l'utilisation de paquets de taille minimale signifie qu'un plus grand nombre de paquets circuleront sur le réseau. Ainsi la largeur de bande utilisée sera plus grande. Ce calcul est obtenu avec les formules suivantes :

$$\text{Nombre de paquets/s} = \frac{80\ \% \times \text{LB du réseau}}{\text{Taille des paquets}}$$

$$\text{Nombre de paquets/s} = \frac{360\ \text{Mbps}}{\text{Charge utile}}$$

$$\text{Taille du paquet} = \sum_{\text{TL, NL et DLL}}^{\text{Entêtes des couches}} + \text{Charge utile}$$

Avec les substitutions appropriées, nous obtenons la formule suivante, qui permet de déterminer la charge utile minimale d'un paquet afin de permettre la transmission d'un signal vidéo numérique requérant 360 Mbps :

$$\text{Charge utile} = \frac{360\ \text{Mbps} \times \text{Entêtes}}{80\ \% \times \text{LB du réseau} - 360\ \text{Mbps}}$$

Tableau 2.3 Largeurs de bande utilisées pour des paquets de tailles minimales permettant la transmission de vidéo numérique de 360 Mbps

Protocole	Largeur de bande (Mbps)	Taille des en-têtes (bits)			Charge utile minimale d'un paquet (bits)	Taille minimale d'un paquet (bits)	Nombre de paquets/sec	LB utilisée (Mbps)
		DLL	NL	TL				
IEEE 802.3	1000	144	160	160	384	848	937500	795
IEEE 1394b	800	192	160	160	664	1176	542169	638

Il est intéressant de remarquer, qu'en raison de la plus grande vitesse d'opération du réseau ***Gigabit Ethernet (IEEE 802.3)***, la charge minimale utile pour transporter de la vidéo numérique représente 45 % de la taille du paquet, alors que celle du réseau ***Firewire (IEEE 1394b)*** est de 56 %. Par conséquent, si les paquets ont la taille minimale possible, un processeur réseau devra traiter dans ce contexte 73 % plus de paquets ***Ethernet*** que de paquets ***Firewire***.

2.2.1.3 Évaluation du nombre d'opérations disponibles par paquet

Au pire des cas, les réseaux transmettront des paquets de tailles minimales. Le processeur réseau doit donc pouvoir traiter près de 940000 paquets par seconde. En considérant une vitesse d'opération du processeur de 200 MHz et une opération par cycle d'horloge, nous obtenons le nombre maximum d'opérations pouvant s'effectuer sur chaque paquet.

$$\text{Nombre maximum d'opérations /paquet} = \frac{\text{Fréquence du processeur} \times \text{Nombre d'opérations par cycle}}{\text{Nombre de paquets/s}}$$

$$= \frac{200\ \text{MHz} \times 1\ \text{opération/ cycle}}{940\ \text{k paquets/s}}$$

$$= 212\ \text{opérations/paquet}$$

La taille de la mémoire interne d'instructions du processeur embarqué (GF) est de 256 mots de 32 bits. Ce choix de la taille a été considéré suite à un premier estimé du nombre

d'instructions requis pour la conversion de protocoles. Ce qui confirme la fréquence de 200 MHz du processeur embarqué.

2.2.1.4 Évaluation de la largeur de bande des bus de communication internes du processeur réseau

Maintenant que nous connaissons quels protocoles sont en mesure de transmettre de la vidéo numérique selon la qualité désirée, nous pouvons évaluer la largeur de bande minimale que le processeur réseau doit posséder afin de ne pas devenir un goulot d'étranglement. La méthode de calcul de la largeur de bande des bus de communication internes du processeur est assez simple. Il suffit de faire la somme de tous les débits entrant et sortant du processeur comme à la figure 2.3.

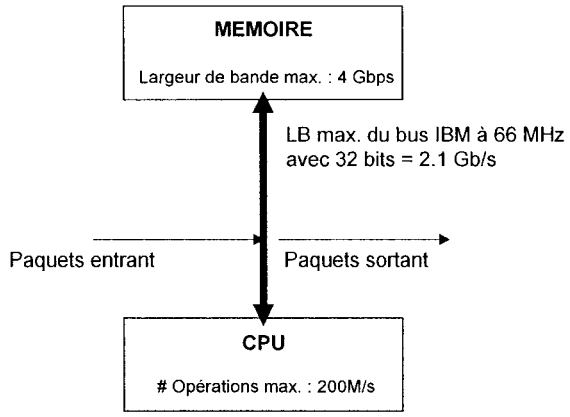


Figure 2.3 Largeur de bande (LB) max. du bus de communication

Il est évident que la réponse est fonction de l'architecture. Autrement dit, si nous avons plus d'un port d'entrée et de sortie, le résultat sera fonction de cette réalité. Dans notre cas, nous avons décidé de nous limiter à un seul port d'entrée et de sortie. La façon la plus conservatrice pour calculer la largeur de bande dans cette circonstance, est de considérer le protocole de plus haut débit (Gigabit Ethernet) avec 795 Mbps comme débit entrant et sortant (voir le tableau 2.3). Nous obtenons donc 1.6 Gbps comme bande

passante minimale des bus de communication sans compter la largeur de bande maximale de la mémoire.

Dans notre architecture, nous avons choisi d'utiliser le mot (32 bits) comme élément de transport des parties d'un paquet. Ainsi, la mémoire d'entrée doit pouvoir être remplie à un rythme de 1 Gbps, soit 31,25 M mots par seconde. De plus, elle doit être lue à un taux équivalent par chacun de ses ports de lecture. Il en résulte que nous avons besoin d'une mémoire possédant une largeur de bande de 120 M mots par seconde ou 4 Gbps. Nous considérons ces performances comme atteignables à l'aide d'une mémoire embarquée multiport. Par exemple, si nous souhaitons qu'il ne faille qu'un cycle d'horloge pour obtenir le résultat d'une lecture, il faut donc que la mémoire fonctionne à 200 MHz comme les autres modules du circuit. Cette fréquence ne constitue pas non plus une difficulté insurmontable pour une mémoire embarquée. Bien entendu, il en est de même pour la mémoire de sortie (figure 2.2), à la différence qu'elle possède un port de moins.
La seule autre source de problème au niveau des communications dans l'architecture se retrouve au niveau des modules qui partagent un même bus de communication. C'est le cas des engins de formatage et du processeur embarqué ARM, qui font offices de maîtres auprès des mémoires esclaves. Il est nécessaire d'évaluer les besoins en largeur de bande de ces éléments afin de choisir un bus de communication approprié.

L'architecture permet la mise en œuvre de plus d'un engin de formatage dans un circuit. La mémoire principale n'offre par contre pas plus d'un port de lecture par l'intermédiaire du bus. Ce port devra donc être partagé entre les engins et le ARM. Si cette mémoire avait été munie de plus d'un port de lecture, il aurait fallu tous les considérer dans le calcul de la largeur de bande. Aussi, la mémoire temporaire offre un port d'écriture et un port de lecture.

Au total, nous avons donc 3 ports reliés à la mémoire principale, soit 16.8 Gbps en ne considérant que les bus de données (3* (4 + 1.6)) Gbps (voir la figure 2.3). Même si nous

avons un nombre **x** d'engins de formatage et un processeur embarqué ARM, il n'est pas utile que le bus possède, pour l'architecture actuelle, une largeur de bande plus grande que ce que peuvent délivrer les mémoires.

En utilisant deux bus distincts pour l'écriture et la lecture mémoire, il est possible de réduire de moitié la largeur de bande du bus de communication qu'utilisent les engins et le ARM.

Tableau 2.4 Largeurs de bande requises par les différents éléments de l'architecture

Élément de l'architecture	Nombre de ports	Largeur de bande (Gbps)
Mémoire multiport principale	4	22.4
Mémoire multiport temporaire	3	16.8
Engins de formatage (lecture)	x	5.6x
Engins de formatage (écriture)	x	5.6x
ARM (lecture)	1	5.6
ARM (écriture)	1	5.6

2.2.2 Protocoles supportés

2.2.2.1 Le modèle OSI

L'organisation des paquets repose généralement sur un modèle en couche comme le modèle OSI. Ce modèle repose sur un principe d'encapsulation. Le paquet est décomposé en 7 couches différentes, comme décrit sur le schéma suivant.

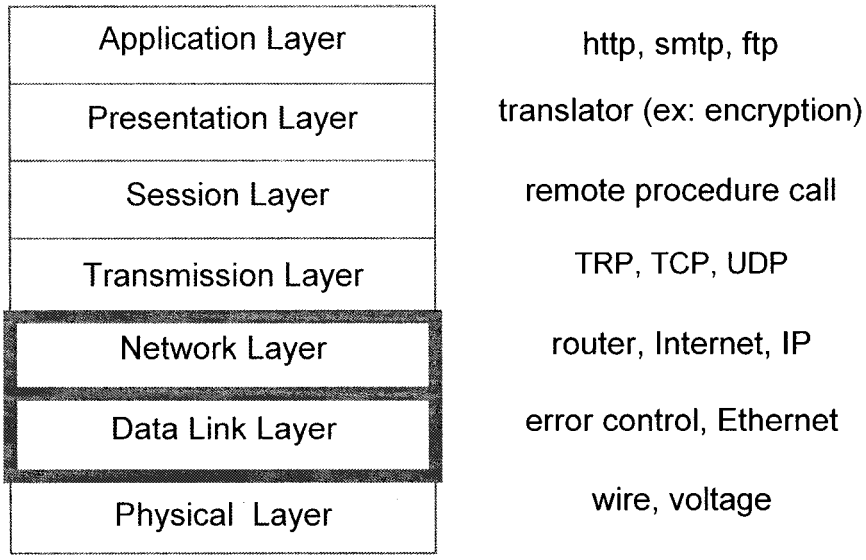


Figure 2.4 Couches du modèle OSI

Chaque couche se distingue par son en-tête et sa queue, le reste étant pour elle, considéré comme charge utile. L'interface physique se charge d'interpréter la couche physique CP. La première version de notre convertisseur traitera les couches DLL et NL. Pour notre architecture, le paquet est donc vu de la façon suivante :

en-tête DLL	en-tête NL	charge utile	queue NL	queue DLL

La conversion n'intervient que sur les en-têtes et queues de chaque couche. La charge utile, qui peut représenter jusqu'à 90 % d'un paquet, ne subit aucune modification. Cette politique de traitement conduit à des gains de performance très importants en comparaison d'une architecture où les unités de traitement manipuleraient non seulement les en-têtes, mais aussi les données.

La première version de notre architecture est en mesure de convertir le protocole IEEE 802.3 (Gigabit Ethernet) [22] vers IEEE1394 (Firewire) [23]. Nous avons aussi exploré les protocoles IPv4 et X25 relatifs à la couche NL ainsi que TCP et UDP de la couche

TL. Quelques données de base sur les protocoles traités sont fournies dans les deux sections suivantes.

2.2.2.2 IEEE 802.3 (Gigabit Ethernet)

Les propriétés de ce protocole sont :

- Taille minimum des trames : 64 octets (DA (*Destination Address*) à FCS (*Frame Check Sequence*) inclusivement)
- Taille maximum des trames : 1518 octets
- Vitesse de transfert : 1000 Mb/s

Le paquet peut-être divisé en plusieurs champs qui sont décrits ci-après :

Tableau 2.5 Description du paquet *Ethernet*

Nom	Taille (bits)	Fonction
Start Frame Delimiter	8 (1 octet)	Toujours 0xAB, bit 0 d'abord
Destination Address	48 (6 octets)	Adresse de destination.
Source Address	48 (6 octets)	Adresse source
Length/type	16 (2 octets)	Taille des données ou indicateur du type de paquet
Data	1500 octets max	Données
Pad	x	Remplissage si la taille de la trame < la taille min. de trame (pour Gb Ethernet, c'est 64 octets)
Frame Check Sequence	32 (4 octets)	CRC
Extension	y	Information concaténée à la fin de la trame. Nous ne l'utiliserons pas ici.

2.2.2.3 IEEE 1394 (Firewire)

Les propriétés de ce protocole sont :

- Taille minimum des trames : 24 octets (DA à FCS inclusivement)
- Taille maximum des trames : 2072 octets, 4120 octets
- Vitesse de transfert : 400 Mb/s, 800 Mb/s

Le paquet peut-être divisé en plusieurs champs qui sont décrits ci-après :

Tableau 2.6 Description du paquet *Firewire*

Nom	Taille (bits)	Fonction
Destination ID	16	Identificateur (adresse) de destination
Transaction label	6	Nom unique donné à chaque transaction.
Retry code	2	Indique si la transaction est une nouvelle tentative après un premier essai manque. Forcé à '01' pour cette implantation (01 = non-supporté).
Transaction code	4	Identifie le type de paquet. Nous nous limitons à un type : le code pour ce type est 7
Priority	4	Fixé à zéro
Source ID	16	Identificateur (adresse) de la source
Response Code	4	Identifie si le paquet est une réponse à une requête précédente.
Reserved	44	Réservé pour expansion future : mettre tout à zéro.
Data Length	16	Taille des données
Extended Transaction Code	16	Pas utilisé dans cette implantation : forcé à 0x0000
Header CRC	32	CRC calculé à partir de l'en-tête
Data	jusqu'à 2048	Bloc de données
Data CRC	32	CRC calculé à partir des données

En résumé, les discussions présentées dans ce chapitre font la synthèse d'une phase de recherche qui a contribué à l'élaboration d'une architecture pouvant servir comme plate-forme SoC. Nous avons aussi présenté diverses notions pertinentes reliées aux protocoles. Celles-ci se révèlent être le véritable cœur du projet. Un aperçu sur les solutions existantes a aussi permis de cerner leurs limitations et de voir dans quel sens devait tendre l'architecture : un des points importants retenus est la flexibilité désirée. Notre plate-forme doit être capable de convertir divers protocoles, selon le programme chargé dans ses mémoires embarquées.

Dans le prochain chapitre, nous présenterons le développement de l'engin de formatage inspiré de modèles de processeurs RISC 32 bits, spécialisés dans la manipulation de données. Cependant, le jeu d'instructions a été adapté pour le rendre spécifique aux besoins de la conversion définis lors de l'étude des algorithmes de conversion des protocoles.

CHAPITRE III

Conception d'un processeur embarqué dédié

Ce chapitre fournit la spécification fonctionnelle pour l'engin de formatage ***GF*** (« General Formator »). Ce module constitue l'unité principale dans le processus de conversion des différents protocoles de communication. En effet, l'engin de formatage est un processeur embarqué capable d'effectuer les instructions d'un programme de conversion contenu dans sa mémoire locale. C'est un module dont les opérations ont été adaptées au processus de conversion de protocoles. Également, nous définirons un plan des vérifications à suivre, ainsi que les différents cas à tester. Nous distinguerons différents cas de test : ceux fondés sur la fonctionnalité, ceux qui exercent les interfaces et ceux qui stimulent les cas limites.

Étant donné les contraintes temporelles imposées par la compagnie Gennum qui parraine ce travail de recherche, la première version du ***GF*** a été simplifiée. L'engin de formatage est un processeur dont les opérations ont été adaptées au processus de conversion; ces opérations se divisent en 5 groupes :

- ***Les opérations arithmétiques et logiques*** :
  - **ADD** : addition
  - **SUB** : soustraction
  - **AND** : ET-logique
  - **OR** : OU- logique
  - **XOR** : OU-Exclusif logique
  - **MSK** : masquage

- ***Les opérations de décalage*** :

  - **SHR** : décalage à droite
  - **SHL**: décalage à gauche

- ***Les opérations de transfert*** :

  - **MOV** : Transfert d'un registre dans un autre.
  - **LD, LDI** : Transfert du contenu d'une adresse dans un registre.
  - **LDB, LDW** : Transfert de certaines parties d'un registre dans un autre.
  - **LDIL, LDIH** : Transfert de certaines parties du contenu d'une adresse dans un registre.

- ***Les opérations de saut*** :

  - **RJMP** : Saut d'une instruction à une autre.
  - **RB0, RB1** : Saut conditionnel d'une instruction à une autre.

- ***Les opérations spéciales*** :

  - **NOP** : Pas d'opération effectuée.
  - **IDLE** : Mode de vieille. (Idle)

3.1 Les entrées/sorties du processeur embarqué

La figure 3.1 illustre toutes les entrées/sorties associées au processeur embarqué, aussi nommé GF.

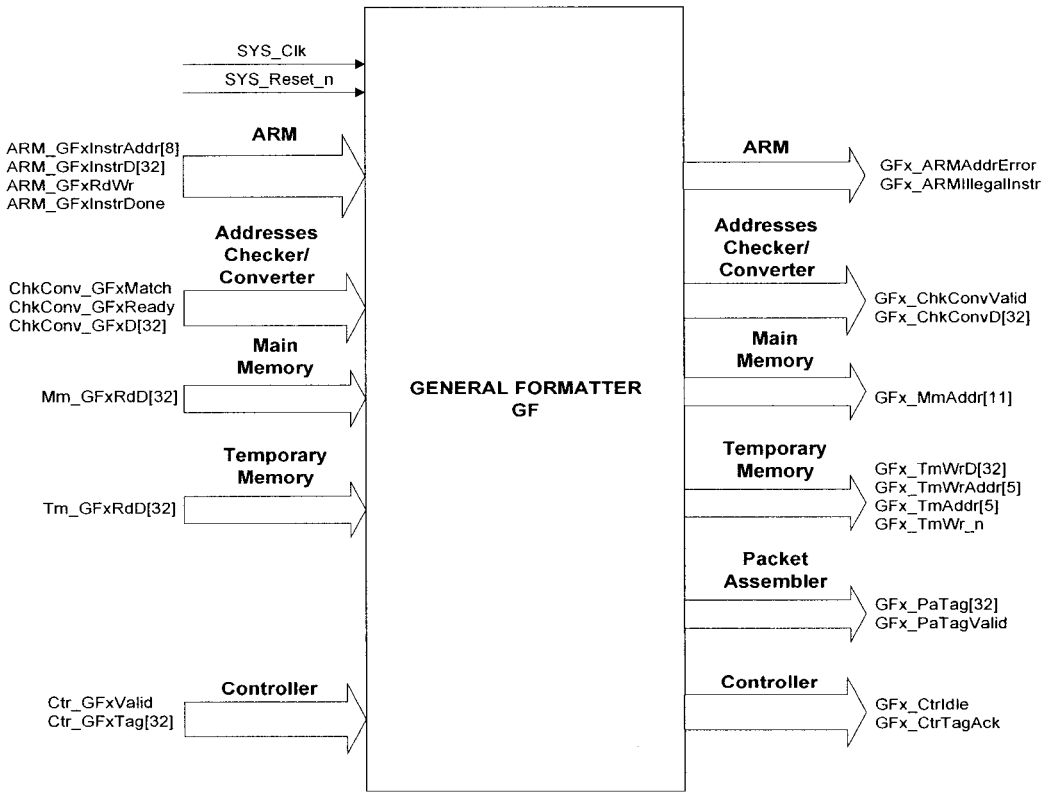


Figure 3.1 Diagramme Bloc de haut-niveau du GF

3.2 Description des signaux d'Entrées/Sorties

Une description détaillée de tous les signaux d'entrées et de sorties est donnée ici :

SYS_clk	horloge du système
SYS_reset_n	reset asynchrone (actif bas)

- ***Signaux d'interface avec le processeur ARM7***

ARM_GFxInstrAddr[7:0]	Bus d'adresses des instructions provenant du processeur ARM
ARM_GFxInstrD[31:0]	Bus de données des instructions provenant du processeur ARM
ARM_GFxInstrDone	Signal indiquant la fin du chargement de la mémoire d'instructions (active haut)
ARM_GFxRdWr	Signal indiquant une écriture dans la mémoire d'instructions du GF (active haut)
GFx_ARMAddrError	Signal indiquant au processeur ARM que l'adresse IP de destination n'existe pas dans la TCAM (*Ternary Content Addressable Memory*).
GFx_ARMIllegalInstr	Signal indiquant au processeur ARM que le GF a reçu une instruction illégale.

- ***Signaux d'interface avec le module du coprocesseur de conversion d'adresses AC***

ChkConv_GFxD[31:0]	Bus de données contenant la donnée traitée par le module « Addresses Checker/Converter »
ChkConv_GFxMatch	Signal indiquant que la donnée est présente dans la TCAM du module « Addresses Checker/Converter » (actif haut)
ChkConv_GFxReady	Signal indiquant que le module « Addresses Checker/Converter » a fini son traitement (actif haut)
GFx_ChkConvD[31 :0]	Bus de données de sortie contenant la donnée à traiter par le module « Addresses Checker/Converter »
GFx_ChkConvValid	Signal indiquant que la donnée provenant de l'engin est disponible (actif haut)

- ***Signaux d'interface avec le module de la mémoire principale***

Mm_GFxRdD[31:0]	Bus de données de sortie lues par l'engin de formatage
GFx_MmAddr[10 :0]	Bus d'adresses pour une lecture en mémoire principale

- ***Signaux d'interface avec le module de la mémoire temporaire***

Tm_GFxRdD[31:0]	Bus de données de sortie de la mémoire temporaire vers l'engin de formatage
GFx_TmAddr[4:0]	Bus d'adresses du mot à lire dans la mémoire temporaire
GFx_TmWr_n	Signal de « write enable » pour commander une écriture dans la mémoire temporaire (actif bas)
GFx_TmWrAddr[4:0]	Bus d'adresses du mot à écrire dans la mémoire temporaire
GFx_TmWrD[31:0]	Bus de données de sortie à écrire dans la mémoire temporaire

- ***Signaux d'interface avec le module du coprocesseur d'assemblage du paquet PA***

GFx_PaTag[31 :0]	Tag indiquant les informations nécessaires pour l'assemblage d'un paquet : il sera envoyé en 3*32 bits
GFx_PaTagValid	Signal indiquant que le traitement sur le paquet identifié par GFx_PaTag[31 :0]

- ***Signaux d'interface avec le module du contrôleur CTR***

Ctr_GFTag[31 :0]	Requête destinée à l'engin de formatage, dans le but d'effectuer le traitement sur le paquet identifié dans ce message

Ctr_GFxValid	Signal indiquant que l'engin doit faire le traitement sur le paquet (actif haut)
GFx_CtrIdle	Signal indiquant au contrôleur que l'engin a fini son traitement (actif haut)
GFx_CtrTagAck	Signal indiquant que le signal Ctr_GFxTag a bien été reçu par l'engin de formatage (actif haut)

3.3 Description des registres

L'engin de formatage contient une banque de registres temporaires (« General Purpose Registers ») au nombre de 16 et large de 32 bits, le registre du compteur de programme (« Program Counter ») et le registre d'états du GF (« Status Register ») dont le détail est donné ci-après :

LSB MSB

C	P	Z	N	V	IDLE	CV	AC	Réservé

SREG[0:31]

Figure 3.2 Registre d'états (SREG)

Tableau 3.1 Description du registre du statut

Nom	Position	Description
C	0	Indicateur de retenue
P	1	Indicateur de parité (1= paire/0= impaire)
Z	2	Indicateur de résultat nul suite à une opération logique ou arithmétique
N	3	Indicateur de résultat négatif suite à une opération logique ou arithmétique
V	4	Indicateur de débordement suite à une opération logique ou arithmétique
IDLE	5	Indicateur de disponibilité du GF
CtrValid (CV)	6	Indicateur d'une requête venant du contrôleur
AC	7	Indicateur que le coprocesseur de conversion d'adresses est disponible

Il est à noter que nous ne pouvons pas utiliser les deux derniers registres de la banque de registres temporaires, soient les registres 14 et 15 qui sont réservés pour une utilisation

future. De plus, les bits 6 et 7 du registre du statut sont reliés directement aux entrées ***Ctr_GFxValid*** et ***ChkConv_GFxReady*** respectivement.

3.4 Détails des opérations de lecture et d'écriture

L'engin de formatage **GF** communique avec plusieurs modules, notamment le processeur **ARM**, le coprocesseur de conversion d'adresses **AC**, la mémoire principale, la mémoire temporaire et le contrôleur **CTR**, via des bus différents. Cette communication ressemble à celle entre un microprocesseur et sa mémoire. Le module concerné est accédé par le **GF** selon les trois bits les plus significatifs de l'instruction. Le tableau 3.2 fournit les codes de sélection des modules pour une lecture ou une écriture.

Tableau 3.2 Tableau illustrant la sélection d'un module par le GF

SelBus[2:0]	Mode	Module sélectionné
000	Lecture	Contrôleur **CTR**
001	Lecture/Écriture	Coprocesseur de conversion d'adresses **AC**
100	Écriture	Coprocesseur **PA**
010	Lecture	Mémoire principale
011	Lecture/Écriture	Mémoire temporaire
101		Aucun
110		Aucun
111		Aucun

3.4.1 Interface entre le GF et le processeur ARM7

Le GF ne possède pas de mémoire cache de données : nous supposons que les temps d'accès aux mémoires dont nous disposons sur notre chip sont suffisamment courts. En revanche, il contient une cache d'instructions de 256 * 32 à l'intérieur de laquelle sera chargé un programme de conversion après chaque remise à zéro de l'ensemble de l'architecture. Lors de la phase de chargement, nous écrivons sur chaque front montant de l'horloge. Le diagramme temporel est donné par la figure 3.3. L'écriture dans la mémoire d'instructions est activée via le signal ***ARM_GFxRdWr***. Par la suite, chaque instruction est chargée dans la mémoire du **GF** jusqu'à la dernière instruction. Quand toutes les

instructions ont été chargées, le signal ***ARM_GFxInstrDone*** est alors, activé pour signifier la fin du chargement. Le début de l'exécution des instructions est activé par une combinaison de deux actions, à savoir une déactivation du signal ***ARM_GFxRdWr*** avec activation du signal ***ARM_GFxInstrDone***. Le signal ***GFx_ARMAddrError*** est activé suite à une recherche négative du coprocesseur de conversion d'adresses (***ChkConv_GFxMatch*** = `0`). D'autre part, le signal ***GFx_ARMIllegalInstr*** est activé suite à une instruction non valide, autrement dit, une instruction qui n'existe pas dans le jeu d'instructions du **GF**.

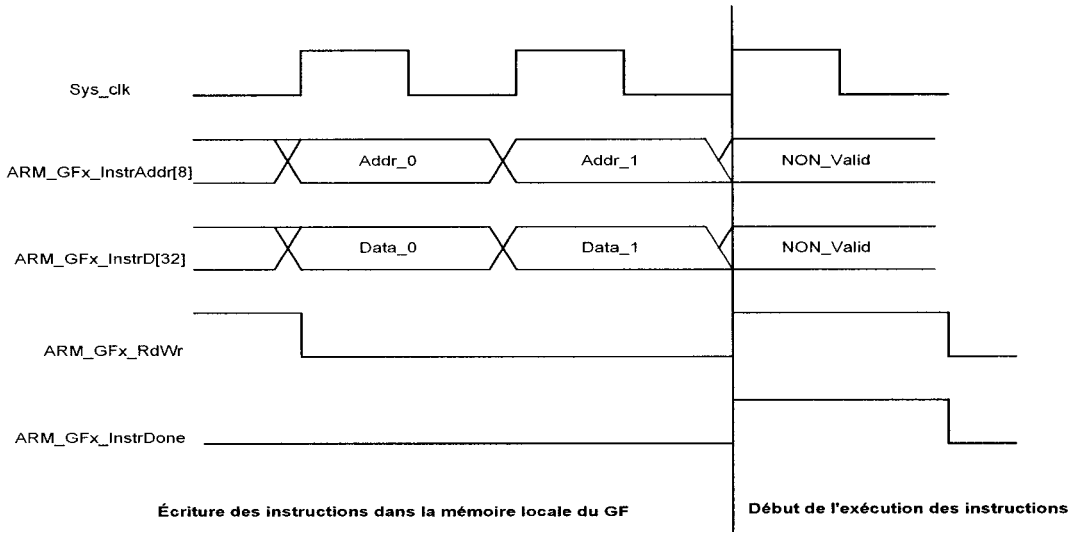


Figure 3.3 Diagramme temporel de l'interface GF-ARM

3.4.2 Interface entre le GF et le coprocesseur de conversion d'adresses AC

Le coprocesseur de vérification et de conversion des adresses assiste l'engin de formatage dans le traitement des en-têtes. Ainsi, le **GF** envoie les données ***GFx_ChkConvD*** à traiter, au coprocesseur de conversion d'adresses, en activant le signal ***GFx_ChkConvValid*** sur le front montant de l'horloge globale du système ***SYS_clk***. Par contre, le **GF** reçoit les données du coprocesseur de conversion d'adresses (adresse IP convertie) avec un signal ***ChkConv_GFxMatch*** indiquant que la donnée était présente

dans la table de correspondance. En même temps, le coprocesseur de conversion d'adresses envoie au **GF** un signal ***ChkConv_GfxReady*** indiquant la fin de son traitement. La figure 3.4 illustre le fonctionnement de l'interface GF-AC.

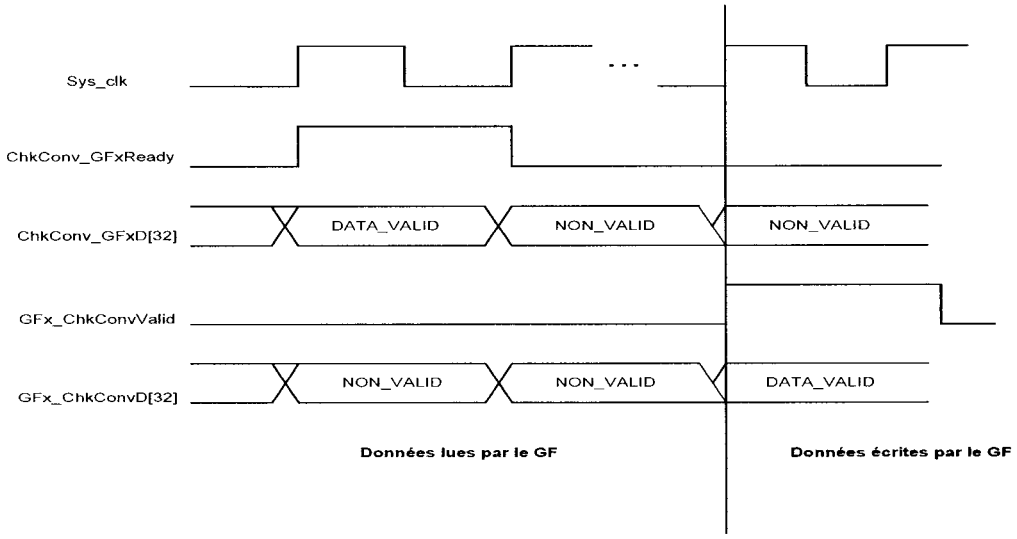


Figure 3.4 Diagramme temporel de l'interface GF-AC

3.4.3 Interface entre le GF et la mémoire principale

Une mémoire principale synchrone a été conçue, basée sur des modèles disponibles dans la bibliothèque ***Synopsys***, disponible à l'École Polytechnique. Ce type de mémoire est aussi supporté par les compilateurs ciblant les FPGA. C'est une mémoire de 2048 mots de 32 bits qui stocke les paquets dans des espaces mémoire de 380 mots (380 mots étant la taille maximale d'un paquet reçu : celle d'un paquet Ethernet); par conséquent, cette mémoire peut contenir au maximum cinq paquets.

La figure 3.5 illustre le fait que les données ***Mm_GFxRdD*** soient synchronisées avec le front montant de l'horloge. Tout changement dans le bus d'adresses ***GFx_MmAddr*** sera suivi d'une lecture de données par le **GF** sur le bus de données ***Mm_GFxRdD***. En effet, une lecture dans la mémoire prend deux cycles d'horloge : un cycle pour envoyer les adresses et un autre pour recevoir les données.

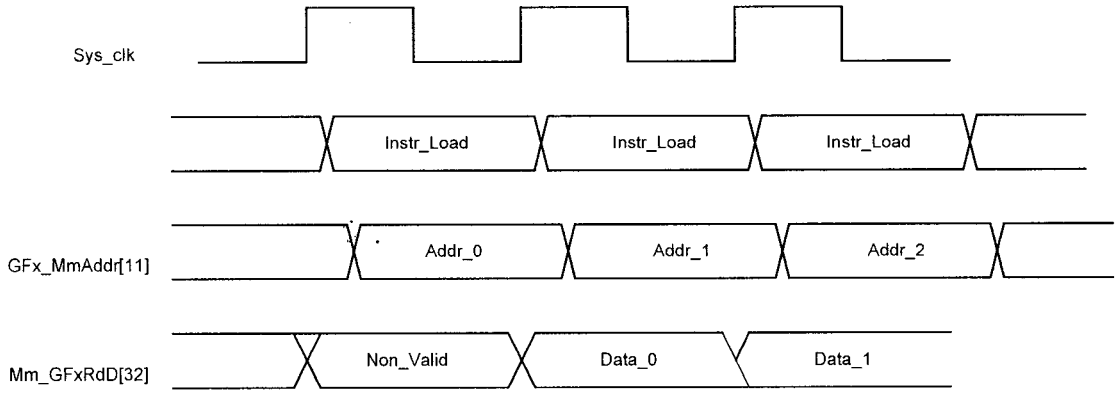


Figure 3.5 Diagramme temporel de l'interface GF-Mémoire principale

3.4.4 Interface entre le GF et la mémoire temporaire

La mémoire temporaire est aussi synchrone. Elle comporte 32 mots de 32 bits afin de contenir les en-têtes des différentes couches traitées par l'engin de formatage **GF**. Cette mémoire stocke les en-têtes traités dans des cases mémoire de 10 mots (la taille maximale des en-têtes étant de 20 octets). Cette mémoire, tout comme la mémoire principale, est basée sur des modèles de la bibliothèque ***Synopsys*** qui supporte, entre autres, les compilateurs FPGA.

En mode lecture, la mémoire temporaire se comporte exactement comme la mémoire principale. La figure 3.6 illustre le fonctionnement de cette mémoire. Tandis qu'en mode écriture, les données sont écrites dans la mémoire suite à une activation du signal de contrôle, en l'occurrence, ***GFx_TmWr_n*** (« write enable »). L'adresse de la case mémoire est envoyée sur le bus d'adresses ***GFx_TmWrAddr***, tandis que la donnée correspondante est mise sur le bus de données ***GFx_TmWrD***. Ceci est illustré dans la figure 3.7.

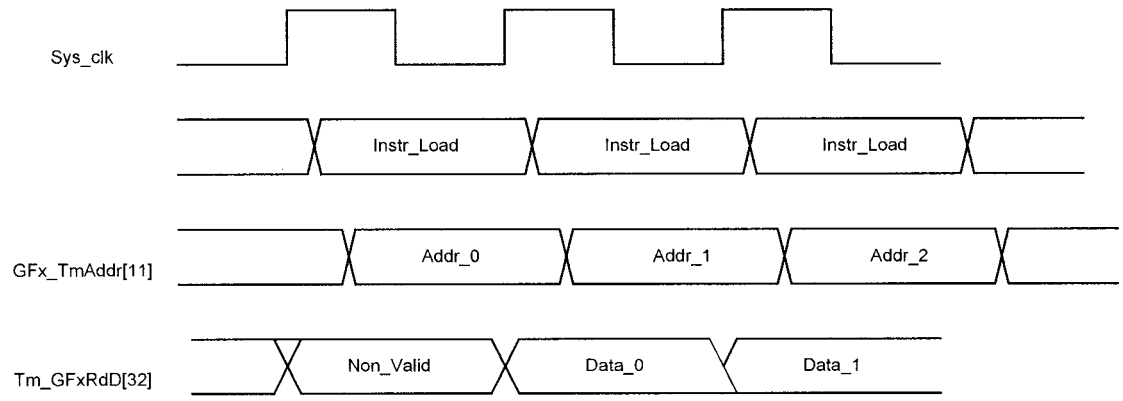


Figure 3.6 Diagramme temporel de l'interface GF-Mémoire temporaire en mode lecture

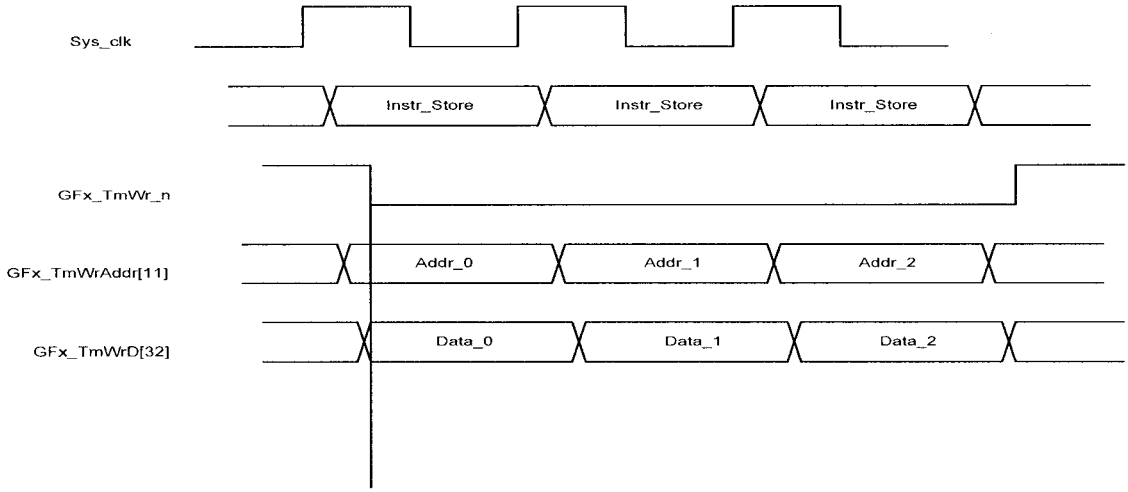


Figure 3.7 Diagramme temporel de l'interface GF-Mémoire temporaire en mode écriture

3.4.5 Interface entre le GF et le coprocesseur d'assemblage de paquets PA

Le coprocesseur d'assemblage des paquets reçoit son information de l'engin de formatage, il rassemble ensuite les données provenant de la mémoire principale et de la mémoire temporaire, avant de transmettre le paquet ainsi reconstitué au coprocesseur à l'interface physique de la sortie.

Le **GF** envoie un signal (étiquette) ***GFx_PaTag*** indiquant les informations nécessaires pour le rassemblage d'un paquet. Ceci sera envoyé en 3*32 bits sur trois fronts montants

d'horloge. Au premier coup d'horloge, le signal ***GFx_PaTagValid*** est envoyé pour indiquer le début du traitement sur un paquet et ceci dure un coup d'horloge. Le mécanisme de communication est illustré dans la figure 3.8.

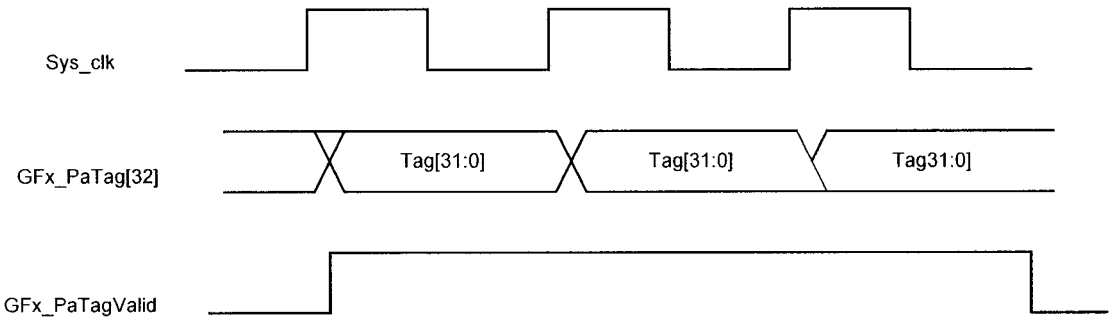


Figure 3.8 Diagramme temporel de l'interface GF-PA

3.4.6 Interface entre le GF et le contrôleur

Le contrôleur du convertisseur de protocoles est un élément charnière de l'architecture dans le sens où il gère l'interaction avec différents modules. Ainsi, il communique avec le gestionnaire de mémoire, le coprocesseur d'identification de protocoles, le registre d'index et l'engin de formatage. Son rôle est la gestion des traitements sur les nouveaux paquets et de leur écriture dans la mémoire principale jusqu'à la fin de leur traitement par l'engin de formatage.

Le contrôleur envoie au **GF** les informations sur le paquet, en l'occurrence, sur le bus de données ***CTR_GFxTag***. Sur le même coup d'horloge, le contrôleur donne au **GF** le signal de départ pour le traitement du paquet ***CTR_GFxValid***. La figure 3.9 illustre la communication entre le **GF** et le contrôleur. Il est à noter que l'engin de formatage envoie à son tour un signal de confirmation de réception ***GFx_CtrTagAck*** et ceci au coup d'horloge suivant la réception. D'autre part, l'engin de formatage avertit le contrôleur de sa disponibilité en activant le signal ***GFx_CtrIdle***.

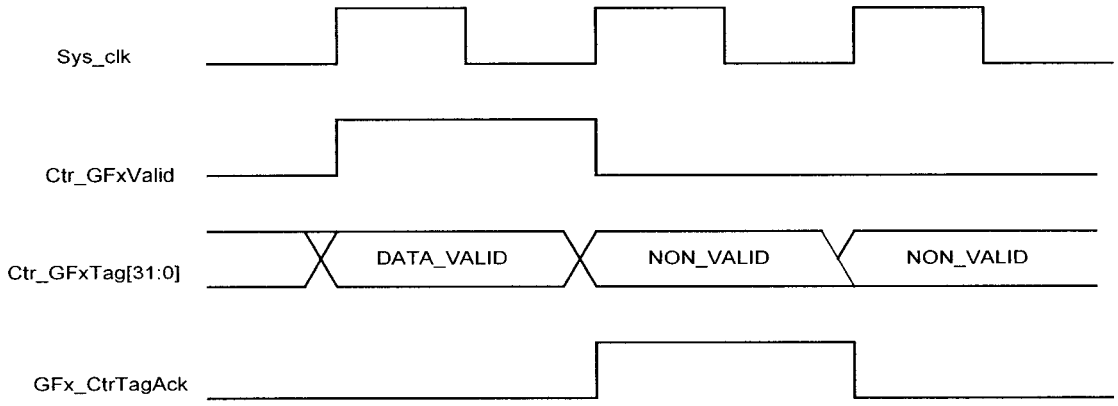


Figure 3.9 Diagramme temporel de l'interface GF-CTR

3.5 Détails du Reset

Dans le cas d'un « reset » global ***SYS_reset_n***, tous les registres de la banque de registres temporaires, le registre d'états, ainsi que le registre du compteur de programme sont mis à zéro (0x00000000). Il est à noter que ce signal est actif bas.

3.6 Détails des opérations arithmétiques, logiques, transfert et de branchement

Les instructions sont codées sur 32 bits. Nous pouvons identifier trois différents types d'instructions dont les formats sont les suivants :

3.6.1 Opérations arithmétiques et logiques

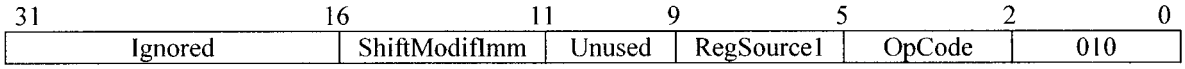
ADD, SUB, AND, OR, XOR, MSK

31 – 23	21 – 19	15 – 11	9 – 5	2 – 0
Ignored	Unused	RegDest	Unused	RegSource2	Unused	RegSource1	OpCode	001

ADI

31 – 27	11 – 9	5 – 2	0
Ignored	AddImm	Unused	RegSource1	OpCode	001

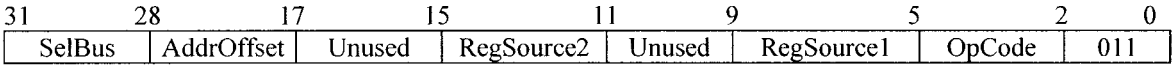
3.6.2 Opérations de décalage



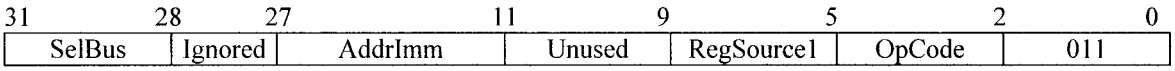
3.6.3 Opérations de transfert

L'engin de formatage peut communiquer avec plusieurs modules externes. Nous avons ainsi dédié un champ de 3 bits (***SelBus***) pour sélectionner le module approprié. Le tableau 3.2 dresse la liste des différents modules pouvant être choisis.

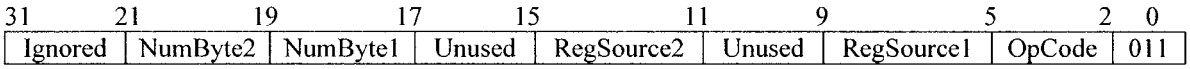
ST, LD



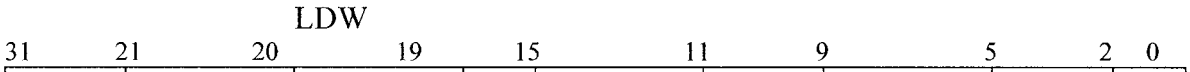
LDI, STI



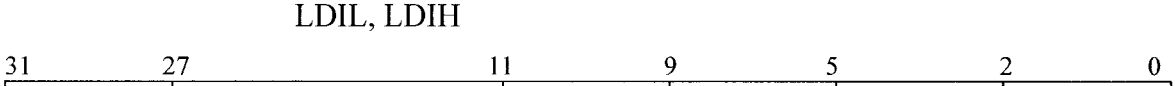
LDB



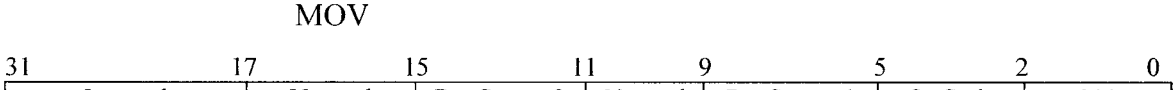
LDW



LDIL, LDIH

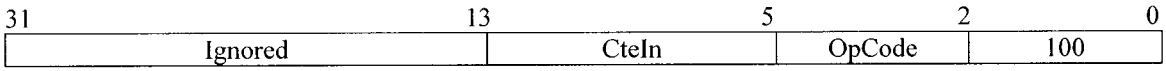


MOV

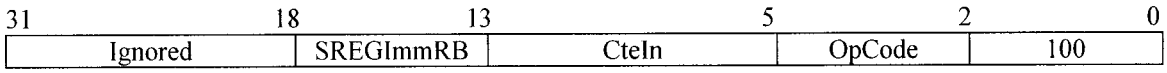


3.6.4 Opérations de saut

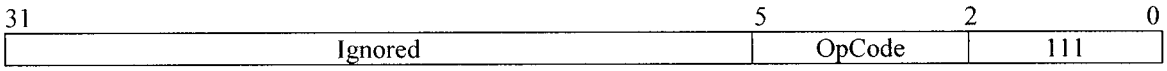
RJMP



RB0, RB1



3.6.5 Instructions spéciales



Selon les instructions réalisées par le Promic de Microlor [43] et les résultats de recherches sur les algorithmes, nous étions en mesure de dresser une liste d'instructions qui pourraient s'avérer utiles pour la manipulation des paquets dans une optique de conversion de protocoles. Le tableau 3.3 résume le jeu d'instructions utilisé pour réaliser l'engin de formatage. Le détail des opérations du jeu d'instructions est donné en annexe A.1.

Tableau 3.3 Le jeu d'instructions utilisé par l'engin de formatage

Nom	Fonction	Instruction	OpGroup	OpCode
Opérations arithmétiques et logiques				
Addition	Ri + Rj => Rk	ADD Ri, Rj, Rk	001	000
Addition avec une constante	Ri + #n => Ri	ADI Ri, #n	001	001
Soustraction	Ri - Rj => Rk	SUB Ri, Rj, Rk	001	010
ET-logique	Ri AND Rj => Rk	AND Ri, Rj, Rk	001	011
OU-logique	Ri OR Rj => Rk	OR Ri, Rj, Rk	001	100
OU-Exclusif	Ri XOR Rj => Rk	XOR Ri, Rj, Rk	001	101
Masquage-logique	Ri & !Rj => Rk	MSK Ri, Rj, Rk	001	110
Opérations de décalage				
Décalage à droite		LSR Ri, #n	010	000
Décalage à gauche		LSL Ri, #n	010	001
Opérations de transfert				
Transfert d'un registre dans un autre	Rj => Ri	MOV Ri, Rj	011	000
Transfert du contenu d'une adresse dans un registre	MS [@] => Ri	LDI Ri, #@, #s	011	001
Transfert du contenu d'une adresse indirecte dans un registre	MS [Rj+o]=> Ri	LD Ri, (Rj), #o, #s	011	010
Transfert de la partie la moins significative du contenu d'une adresse dans un registre		LDIL Ri, #n	011	011
Transfert de la partie la plus significative du contenu d'une adresse dans un registre		LDIH Ri, #n	011	100
Transfert d'un octet d'un register dans un autre		LDB Ri, #n, Rj, #m	011	101
Transfert d'un mot d'un register dans un autre		LDW Ri, #n, Rj, #m	011	101
Rangement du contenu d'un registre	Ri => MS[@]	STI Ri, #@, #s	011	110
Rangement indirect du contenu d'un registre	Ri => MS[Rj+o]	ST Ri, (Rj), #o, #s	011	111
Opérations de saut				
Saut relatif	PC + #d => PC	RJMP #d	100	000
Branchement conditionnel si le code est égal à 0	If (SREG (n) = 0) (PC) + #d => PC	RB0 #d, #n	100	101
Branchement conditionnel si le code est égal à 1	If (SREG (n) = 1) (PC) + #d => PC	RB1 #d, #n	100	110
Instructions spéciales				
Pas d'opérations		NOP	111	000
Mode de vieille		IDLE	111	001
Reset		RSTIDLE	111	010

3.7 *Méthodologie de vérification*

La méthodologie de vérification que nous avons utilisée, est basée sur les méthodes et les techniques exposées dans [7] et [17]. La figure 3.10 présente le processus de vérification que nous appliquerons dans le modèle sous vérification, en occurrence, le processeur embarqué.

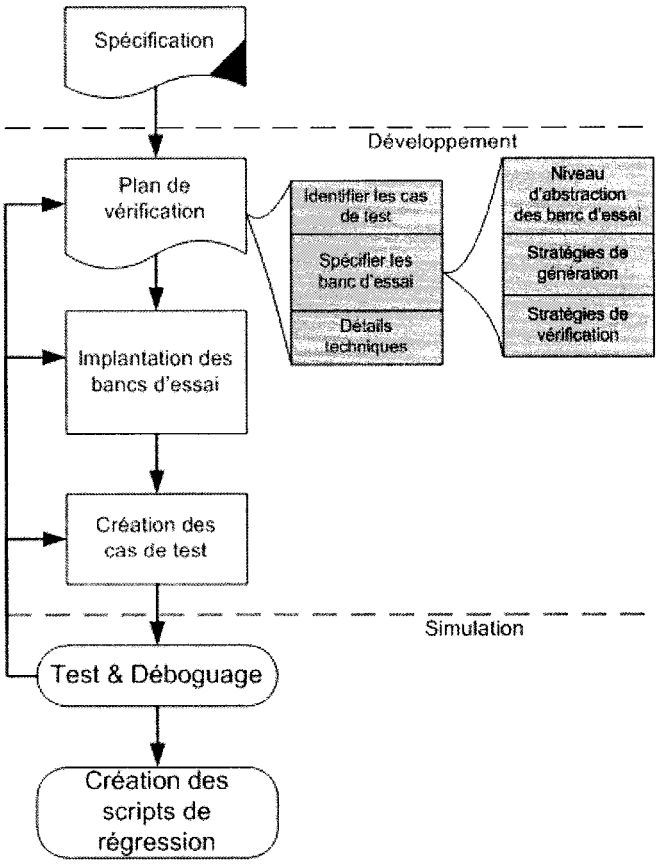


Figure 3.10 : Méthodologie de vérification

La première étape de la méthodologie de vérification est l'étude des spécifications du design. Cette étude permet de comprendre le design, mais aussi d'identifier les lacunes dans la spécification et ainsi, d'effectuer des corrections avant même d'avoir commencé les tâches de vérification. Ensuite, nous avons élaboré un plan de vérification qui sera présenté à la section suivante. Le plan de vérification est la spécification de la vérification. Il doit être conçu correctement afin de pouvoir effectuer une vérification efficace qui permettra la détection d'erreurs et ainsi augmenter le degré de confiance dans le design. Le plan de vérification est divisé en différentes sections qui sont spécifiées dans le tableau 3.4.

Tableau 3.4 Détails du plan de vérification

Étape		Description
Identifier les cas de test		L'identification des cas de test permet d'identifier tout ce qui doit être vérifié dans le design. Cette étape est la plus importante de la méthodologie de vérification car elle détermine implicitement les cas qui ne seront pas vérifiés.
Spécifier les bancs d'essai		Les bancs d'essai servent à vérifier ce qui a été identifié dans l'étape d'identification des cas de test. Ainsi, les bancs d'essai doivent être spécifiés de façon à pouvoir réaliser tous les cas de test. Il peut être nécessaire de spécifier plusieurs bancs d'essai pour un design afin de réaliser tous les cas de test. Les trois points suivants sont les méthodes utilisées afin d'implanter les bancs d'essai. Ces points sont fortement influencés par le choix des outils utilisés.
	Niveau d'abstraction	Nous déterminons le niveau d'abstraction du banc d'essai. À un extrême, nous pouvons travailler au niveau binaire (bas niveau). À l'autre extrême nous pouvons travailler à un haut niveau via des structures de données complexes.
	Stratégie de génération	Nous déterminons comment les séquences de vecteurs de test : déterministe ou pseudo-aléatoire, seront générées. Les vecteurs peuvent être générés dans le banc d'essai ou à l'externe (chargement à partir du fichier).
	Stratégie de vérification	Nous déterminons comment les réponses du modèle sous vérification seront testées. Elles peuvent être vérifiées dans le banc d'essai (auto-vérifiant) ou à la suite de la simulation.
Détails Techniques		Les détails techniques sont toutes les informations que nous voulons inclure dans le plan de vérification (structure de répertoire, scripts, outils, licences, etc.).

La prochaine étape de la méthodologie est l'implantation des bancs d'essai. Dans le cadre de ce projet, nous avons utilisé le langage HVL (Hardware Verification Language) ***e*** avec l'outil Specman Elite™. Le langage ***e*** est un langage dédié à la réalisation de bancs d'essai. L'utilisation de ce langage permet d'implanter des bancs d'essai à un haut niveau d'abstraction et ainsi, facilite la réalisation de bancs d'essai auto-vérifiant. De plus, Specman Elite™ fournit un générateur pseudo-aléatoire pouvant être paramétré par des contraintes de génération. Celles-ci permettent d'adopter une stratégie de génération de vecteurs de test pseudo-aléatoire évoluée. La méthodologie de conception de banc d'essai sera basée sur la méthodologie proposée dans [34].

Cette méthodologie utilise les concepts orientés objet et orientés aspect afin de concevoir un banc d'essai. Un objet est une structure de données pouvant contenir à la base des champs de données et des fonctions. Un aspect définit un comportement ou une fonctionnalité qui affecte plusieurs objets d'un système. Ainsi, en effectuant une modélisation orientée objet d'un banc d'essai, nous obtenons un banc d'essai composé d'une hiérarchie d'objets. Ensuite, si les fonctionnalités qui sont communes à tous les

objets, sont isolées dans un aspect, cela facilite la réutilisation et la maintenance de certaines parties d'un banc d'essai. En fait, lorsqu'une fonctionnalité change, comme elle a été isolée dans un aspect, les modifications nécessaires sont seulement effectuées sur l'aspect spécifique et non sur tous les objets affectés par cet aspect. Par exemple, dans un banc d'essai modélisé en objet, plusieurs classes auront des connexions et des interactions avec le DUT (*Design Under Test*). Si nous isolons toutes les interactions avec le DUT dans un seul aspect, il est maintenant plus simple d'actualiser le banc d'essai lorsqu'un changement se produit aux interfaces du DUT. Les catégories d'aspects dédiées à la conception de bancs d'essai sont décrites au tableau 3.5.

Tableau 3.5 Description du partitionnement par aspect

Aspect	Description
Base	Définit la base de chaque objet.
HDL	Définit les connexions et les interactions avec le modèle HDL.
Logique	Définit les connexions et les interactions entre les objets.
Configuration	Définit les cas de test à appliquer sur le modèle sous vérification.
Couverture	Définit des modèles de couverture fonctionnelle permettant d'avoir des résultats quantitatifs sur la simulation.

Par la suite, les cas de test définis dans le plan de vérification seront implémentés et exécutés. Dans la méthodologie de partitionnement par aspects, les cas de test entrent dans la catégorie Configuration.

Lorsque toutes les étapes de développement des bancs d'essai seront réalisées, la prochaine étape sera le test et le débogage. Dans cette étape, il faut simuler tous les cas de test définis et corriger les erreurs détectées. Il est à noter que les erreurs peuvent provenir autant du design que du banc d'essai. Il faut donc être critique lors de l'analyse des résultats obtenus. Lorsque le design sous vérification a passé tous les cas de test définis, il faut créer des scripts de régression. Ces scripts serviront à exécuter en séquence tous les cas de test du plan de vérification. Ainsi, avec ces scripts, la suite de tests de régressions pourra être exécutée à chaque modification dans le design afin de pouvoir déterminer si la modification n'a pas introduit une erreur.

La méthodologie de vérification que nous allons utiliser est définie en séquence, mais le processus est itératif. Ainsi, il n'y a pas de problèmes à revenir sur des étapes précédentes.

3.7.1 Cas de test

Dans cette section, les cas de test à appliquer sur les « designs » sous vérification sont définis. La définition de cas de test détermine ce qui sera vérifié dans les « designs » et la méthode de test qui sera utilisée afin de vérifier chaque cas. La méthodologie utilisée afin de définir les cas de test, propose trois catégories de tests :

- ***Tests par fonctionnalité***

  Ils servent à couvrir toutes les fonctionnalités du « design » listées dans la spécification du design. Ainsi, l'exécution de tous les cas de test définis dans cette catégorie, doit permettre de tester tout ce que le design doit être en mesure de faire.

- ***Tests par interface***

  Ils doivent valider les interactions et les opérations pouvant être effectuées sur chaque interface du design. La validation de ces cas de test permettra d'assurer le bon fonctionnement du design avec l'environnement dans lequel il devra être intégré.

- ***Tests par cas limite***

  Ils sont dépendants de l'architecture interne du design. Ainsi, en effectuant une analyse de l'implantation du design, il faut cibler les limites du design. Les tests

par cas limite servent à cerner des erreurs potentielles pouvant provenir de l'implantation, et que les tests par fonctionnalité et par interface peuvent avoir de la difficulté à identifier.

Les cas de test seront définis sous forme de tableau tel que défini ici :

#	Priorité	Description	Méthode de test

Le premier champ du tableau définit un numéro unique pour chaque cas de test dédié à un module précis. Ensuite, la priorité définit l'ordre dans lequel les cas de test doivent être exécutés. Cet ordre est important, car certains cas de test devront prendre pour acquis que certains éléments ont déjà été vérifiés. Par exemple, il est essentiel de vérifier que le *Reset* d'un système fonctionne avant que nous puissions vérifier toutes autres fonctionnalités d'un système.

3.7.2 Cas de test du processeur embarqué

Les deux tableaux ci-dessous font état des différents cas de test qui seront effectués sur le processeur embarqué. Le tableau 3.6 élabore les différents tests pour vérifier les fonctionnalités du GF, tandis que le tableau 3.7 énumère les tests qui serviront à la vérification des interfaces reliant le GF au reste des modules de l'architecture.

Tableau 3.6 GF - Tests par fonctionnalité

#	Priorité	Description		Méthode de test
GF_1	5	Le chargement des instructions dans la mémoire d'instructions du GF doit s'effectuer correctement.		Des séquences de mots de 32 bits seront envoyées aléatoirement dans l'interface ARM et pour chaque mot envoyé, il sera vérifié automatiquement si le mot a bien été écrit dans la mémoire du GF.
GF_2		Le GF exécute correctement toutes les instructions de son jeu d'instructions et le registre d'états et le registre compteur du GF sont correctement modifiés.		L'émulation de la mémoire principale et de la mémoire temporaire servira de base à la vérification du jeu d'instructions. Un module de vérification permettra de valider l'exécution des instructions.
GF_2.1	8	Instructions de transfert (8)		Pour les instructions de transferts, les instructions devront être validées avec un Registre pour lire et écrire respectivement dans les mémoires principale et temporaire. Ensuite, des séquences d'instructions seront envoyées dans la mémoire d'instructions du GF afin de vérifier que toutes les instructions de transferts sont correctement exécutées. Une image externe du fichier de registre (14 registres) et la séquence d'instructions placée en mémoire sera conservée afin d'auto-vérifier les réponses du DUT.
		MOV	Reg à Reg	
		LDI	GFBus (addr. imm11) à Reg	
		LD	GFBus (addr. Reg + imm11) à Reg	
		LDIL	Imm16 à Reg (lower [15 :0])	
		LDIH	Imm16 à Reg (upper [32 :16])	
		LDB	Échange de bytes entre 2 Reg	
		LDW	Échange de words entre 2 Reg	
		STI	Reg à GFBus (addr. imm1)	
		ST	Reg à GFBus (addr. Reg + imm11)	
GF_2.2	7	Instructions arithmétiques et logiques		Des séquences d'instructions arithmétiques et logiques seront générées et ensuite placées dans la mémoire d'instruction du GF. Les données servant à effectuer les opérations générées sont placées dans la mémoire principale et les résultats seront recueillis dans la mémoire temporaire. Donc, en ayant la séquence d'instructions injectée et les données de base aux opérations demandées, nous pourrons déterminer ce qui devra se retrouver dans la mémoire temporaire à la fin de l'exécution de la séquence d'instruction. Certaine instruction modifie le registre d'états du GF, ainsi l'exécution de toutes les instructions en combinaison avec tous les états du registre d'états est nécessaire.
		ADD	Rdest = Ri + Rj (sans retenue)	
		ADDI	Rdest = Ri + imm16	
		SUB	Rdest = Ri - Rj	
		AND	Rdest = Ri and Rj	
		OR	Rdest = Ri or Rj	
		XOR	Rdest = Ri xor Rj	
		MSK	Rdest = Ri and (not Rj)	
		LSR	Ri = Ri >> imm5 (décalage droite)	
		LSL	Ri = Ri << imm5 (décalage gauche)	
GF_2.3	9	Instructions de branchement		Des séquences d'instructions seront

		RJMP	Saut relatif (imm8)	générées et ensuite elles seront placées dans la mémoire d'instructions. Les instructions générées devront tenir compte de la taille de la mémoire d'instructions qui s'élève à 256 mots. Ainsi, il ne devra pas être permis de générer des instructions de branchement dont le saut dépasse les capacités de la mémoire. Les instructions de saut ont en fait, une influence sur le registre compteur du GF. Ainsi, la vérification de l'exécution correcte de ces instructions sera effectuée en examinant les changements sur le registre compteur du GF. De plus, en fonction du registre d'états, certaines instructions de saut seront ou ne seront pas exécutées. Il faudrait donc vérifier les instructions de branchement en combinaison avec le registre d'états.
		RB0	Saut relatif (imm8)si un bit spécifique du registre de statut est à 0	
		RB1	Saut relatif (imm8)si un bit spécifique du registre de statut est à 1	
GF_2.4	10	Instructions spéciales		La vérification de ces instructions sera effectuée en combinaison avec les autres instructions du jeu d'instructions, précédemment vérifiées.
		NOP	Pas d'opération	
		IDLE	Pas d'opération et le GF est en mode ILDE	
		RSTIDLE	Pas d'opération, le GF est en mode ILDE et le registre compteur n'est pas incrémenté.	
GF_3	14	Le GF est capable de lire ou d'écrire sur tous ses bus de données.		Les instructions de transfert seront utilisées afin de travailler avec tous les bus du GF.
GF_3.1		Bus MM (Lecture)		Les instructions LDI, LD, ST, STI sont les instructions interagissant avec les bus du GF. Ainsi toutes ces instructions devront être utilisées sur tous les bus du GF.
GF_3.2		Bus TM (Écriture)		
GF_3.3		Bus CTR (Lecture)		
GF_3.4		Bus AC (Lecture/Écriture)		
GF_3.5		Bus PA (Écriture)		
GF_4	15	GF est capable d'exécuter des programmes complexes.		Des séquences d'instructions, comprenant tous les types d'instructions, seront générées. Les données qui alimenteront ces séquences seront placées dans la mémoire principale et les résultats seront recueillis dans la mémoire temporaire. Les résultats des séquences d'instructions (petit programme) seront vérifiés automatiquement en effectuant une exécution externe des séquences d'instructions.
GF_5	16	Tous les types légaux d'en-têtes Firewire doivent être formatés correctement en en-têtes		L'algorithme de conversion d'en-têtes FireWire à Ethernet sera

		Ethernet par le GF en utilisant l'algorithme de conversion de l'en-tête Firewire à Ethernet.	chargé dans la mémoire d'instructions du GF. Ensuite, dans une approche à haut niveau d'abstraction, des en-têtes Firewire seront envoyés dans le GF et les en-têtes formatés au format Ethernet seront recueillis en sortie. Finalement, nous vérifierons que la conversion a bien été effectuée. Cela implique que tous les modules interagissant aux interfaces du GF devront être émulés et synchronisés afin de produire l'environnement d'opérations réel du GF.

Tableau 3.7 GF - Test par interface

#	Priorité	Description	Méthode de test		
GF_6	1	Reset	Assertions. Voici les règles devant être respectées :		
			1	Lorsque le Reset est actif	
					Le registre compteur est remis à zéro
					Le registre de statut est remis à zéro
					Les registres généraux sont remis à zéro
GF_7	2	Interface ARM - GF. Il est à noter que cette interface ne respecte en rien le protocole de communication AMBA (AHB) qui devrait être utilisé pour la communication avec un ARM. En fait, cette interface dédiée à la première itération du convertisseur et devra complètement être refaite afin de pouvoir accomplir sa tâche réelle qui serait de travailler avec un ARM.	Assertions. Voici les règles devant être respectées :		
			1	Lorsque le signal ARM_GFx_RdWr est à '0' (mode écriture d'instruction du GF)	
					Le GF doit être en NOT-IDLE
					ARM_GFx_Instruc[32 :0] doit être écrit à l'adresse ARM_GFx_Addr[7 :0] dans la mémoire d'instructions interne du GF
			2	Lorsque la fin des écritures d'instructions arrive (Le signal ARM_GFx_Inst_done est actif et le signal ARM_GFx_RdWr est à '1'	
					Au prochain cycle, le GF ne doit plus être en mode IDLE et il doit lire l'instruction de l'adresse 0 de la mémoire d'instructions.
GF_8	11	Interface CTR - GF Le *Contrôleur* et le GF communiquent ensemble afin que le contrôleur puisse passer une étiquette au GF. Cette dernière indique au GF qu'il doit convertir un en-tête Firewire à un en-tête Ethernet	Assertions Voici les règles devant être respectées :		
			1	Lorsque le GF est en mode NOT-IDLE	
					Le CTR ne doit pas activer le signal Ctr_GfxValid
			2	Lorsque le GF est IDLE et que le CTR active le signal Ctr_GfxValid	
					Le mot sur Ctr_GfxTag est une étiquette valide.
					Le GF active le signal GFx_CtrTagAck pour indiquer qu'il a pris le tag..
					CTR désactive le signal Ctr_GfxValid au cycle suivant l'assertion du signal GFx_CtrTagAck par le GF

GF_9	3	Interface MMem - GF Mmem est seulement utilisé pour effectuer des lectures. Mmem est une mémoire asynchrone. C'est-à-dire que l'adresse est placée sur un cycle et MMem retourne le mot de la mémoire sur le même cycle	Assertions Voici les règles devant être respectées :		
			1	Lorsque GF place une adresse sur GFx_MmAddr	
					Mmem retourne le mot de mémoire correspond à l'adresse sur le même cycle d'horloge.
GF_10	4	Interface TMem – GF TMem est seulement utilisé pour effectuer des écritures. TMem est une mémoire asynchrone. C'est-à-dire que l'adresse et le mot à écrire sont placés sur un cycle et TMem écrit le mot à l'adresse spécifiée sur le même cycle	Assertions Voici les règles devant être respectées :		
			1	Lorsque que le signal GFx_TmWr_n est actif	
					Tmem lit l'adresse sur GFx_TmAddr et le mot sur GFx_TmWrD.
					Tmem écrit le mot dans la mémoire temporaire à l'adresse spécifiée
GF_11	12	Interface AC – GF Le GF envoie une adresse FireWire au AC et le Ac l'adresse Ethernet correspondante	Assertions Voici les règles devant être respectées :		
			1	Lorsque le signal GFx_ChkConvValid est actif pendant un cycle	
					L'adresse FireWire est valide sur GFx_ChkConvD
			2	Lorsque le signal ChkConv_GFxREady est actif pendant un cycle	
					L'adresse Ethernet est valide sur GfxChkConv_GFxD
GF_12	13	Addr_Conv_Match - ARM Lorsque AC ne réussit pas à trouver une adresse Ethernet correspondante à une adresse FireWire, AC envoie un signal qui doit être transmis à l'Interface ARM.	Assertions Voici les règles devant être respectées :		
			1	Lorsque le signal CheckConv_GFx_match est asserté	
					Le signal GFx_ARMAddrError doit être asserté un cycle d'horloge après.

3.7.3 Un condensé des fonctionnalités du GF

Les principales fonctionnalités du GF sont résumées dans le tableau 3.8.

Tableau 3.8 Résumé des fonctionnalités du GF

1	Le GF est un processeur dédié à la manipulation de paquets. Précisément, le GF convertit les en-têtes d'un protocole vers un autre protocole.		
2	Il comprend un jeu de 27 instructions codé sur 32 bits.		
3	Il possède une mémoire d'instructions interne pouvant contenir 256 instructions.		
4	Il possède un registre d'états et un registre compteur interne.		
5	Il possède un fichier de 16 registres (32 bits) dont les registres 0 à 14 sont accessibles par l'usager.		
6	Il communique avec l'extérieur par l'intermédiaire de 6 interfaces		
	ARM	Pour le chargement d'instructions de sa mémoire d'instructions.	
	CTR	Pour recevoir une étiquette provenant du contrôleur indiquant au GF le traitement à effectuer sur une en-tête d'un paquet.	
	MM	Pour lire dans la mémoire principale afin d'avoir accès aux mots composants l'en-tête d'un paquet à traiter.	
	AC	Pour avoir accès au module de conversion d'adresses. (Firewire à Ethernet)	
	TM	Pour écrire dans la mémoire temporaire afin de placer l'en-tête convertie dans cette mémoire.	
	PA	Pour envoyer une étiquette à l'assembleur de paquets afin qu'il puisse assembler l'en-tête convertie d'un paquet avec le chargement de ce même paquet	

Le banc d'essai du GF utilise une approche boîte grise. C'est-à-dire que le banc d'essai a été conçu en ayant une certaine connaissance de la structure interne du DUT, mais le banc d'essai n'imposera pas de valeur sur des signaux internes du « design ». L'architecture du banc d'essai utilisée pour la vérification du GF est illustrée à la figure 3.11.

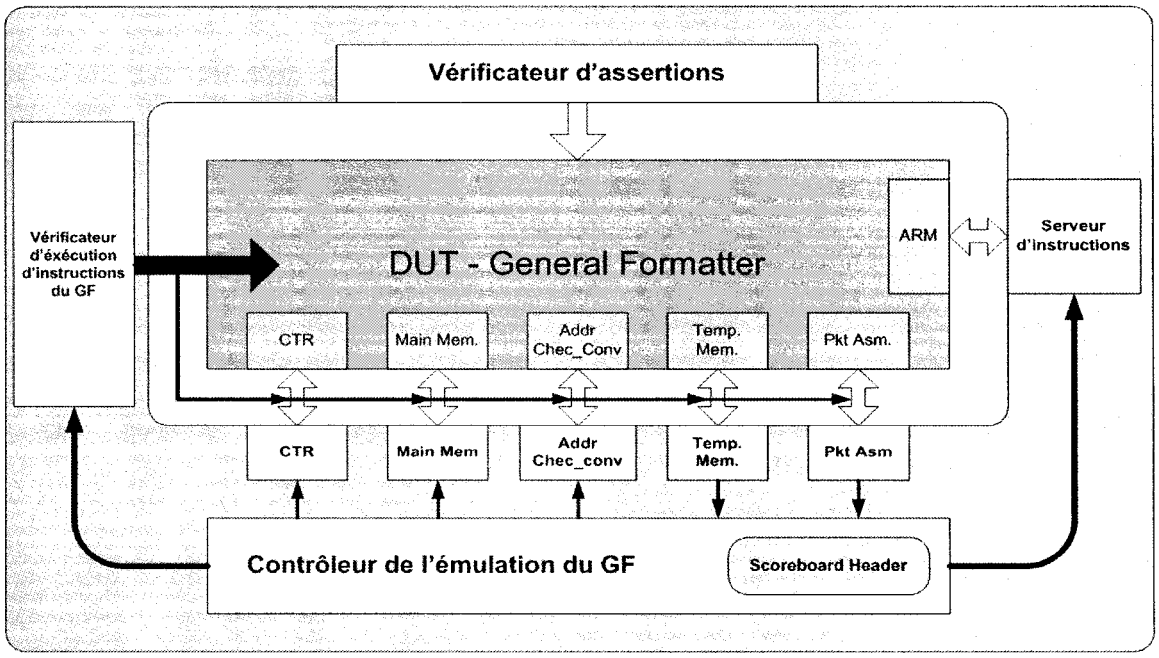


Figure 3.11 Schéma du banc d'essai du GF

Au centre de la figure 3.11, nous retrouvons le DUT et chacune de ses interfaces est illustrée par une boîte. L'ensemble des autres boîtes sont des classes composant l'environnement de vérification. Sur le schéma, seulement les classes de type émulation et de type vérification sont illustrées. Parmi les autres classes qui ne sont pas illustrées sur le schéma, nous retrouvons les classes modélisant :

- Le jeu d'instructions du GF;
- Les paquets FireWire et Ethernet;
- Les étiquettes transigeant entre le contrôleur et le GF et entre le GF et l'assembleur de paquets.

Pour l'émulation de l'environnement du GF, une série de classes a été définie. Pour chaque interface du GF, une classe a été définie afin de pouvoir émuler le comportement à l'interface du GF. Chaque classe émulant l'interface du GF s'occupe de travailler avec les signaux du GF définis pour chaque interface et elle respecte les protocoles définis entre les différents modules du système de conversion de protocoles. Ensuite, une classe ***contrôleur de l'émulation*** est définie, afin de contrôler le comportement des classes aux interfaces du GF. Le contrôleur d'émulation s'occupe de générer les séquences de vecteurs de tests. De plus, il s'occupe d'envoyer les données générées aux classes interfacant avec le DUT. Également, il s'occupe de recueillir les réponses par ces mêmes classes aux interfaces du DUT. Par exemple, afin de créer une séquence d'instructions implantant un cas de test, le contrôleur générera une séquence de 256 instructions (la taille de la mémoire d'instructions). Ces instructions seront chargées dans le DUT via la classe ***serveur d'instructions***.

De plus, afin de pouvoir prédire les résultats du test, le contrôleur générera aussi un fichier contenant les valeurs de 14 registres (14 vecteurs de 32 bits) afin de définir le contenu des registres internes du DUT. Pour placer ces vecteurs dans les registres internes du DUT, il est nécessaire d'utiliser la classe ***émulation de la mémoire***

principale. Ainsi, le contrôleur placera les 14 vecteurs de 32 bits dans la mémoire principale du banc d'essai. Dans la séquence de 256 instructions générées, les 62 premières instructions seront des chargements immédiats (instructions LDI) dans la mémoire principale afin de charger les registres internes. Afin de recueillir l'état des registres internes, les 14 registres devront être ressortis du DUT par l'intermédiaire de la mémoire temporaire. Donc, il faudra aussi que, dans la séquence de 256 instructions, les 62 dernières instructions soient des instructions de rangement (STI) vers la mémoire temporaire. Ainsi, le contrôleur d'émulation pourrait récupérer les registres modifiés provenant du DUT. Alors, entre les 62 instructions de chargement et les 62 instructions de rangement, d'autres séquences d'instructions seront possibles en déterminant un ensemble de contraintes de génération ciblant un cas de test défini précédemment. L'écriture ou la lecture sur les autres interfaces externes du DUT telles que l'interface CTR et l'interface de l'assembleur de paquets est aussi sous la gérance du contrôleur d'émulation.

La création de scénarios générant des séquences d'instructions aléatoirement, sert seulement à explorer l'étendue des possibilités pouvant être rencontrées par le GF et augmenter le degré de confiance dans le GF. Cependant, il est aussi nécessaire de tenter de stimuler le GF dans son contexte d'utilisation réel. C'est pourquoi, le contrôleur d'émulation du GF permet aussi de générer des paquets ***IEEE1394***. Si nous nous situons à un très haut niveau d'abstraction, dans le cadre de la première itération du convertisseur de protocole, le but ultime du GF est de prendre en entrée des en-têtes ***IEEE1394*** et répondre en sortie avec des en-têtes ***Ethernet***. La conversion est réalisée en utilisant un algorithme de conversion spécifié avec le langage assembleur du GF. Donc, pour émuler ce scénario, le contrôleur d'émulation doit charger l'algorithme de conversion dans le DUT. Cela est réalisé par le serveur d'instructions qui est capable de lire les fichiers binaires produits par le compilateur d'assembleur du GF. Ensuite, le contrôleur d'émulation gérera et placera un paquet ***IEE1394*** dans la classe d'émulation de la mémoire principale. Il faudra aussi envoyer dans la classe d'émulation du convertisseur

d'adresses, l'adresse de destination du paquet ***IEEE1394*** et une adresse de destination ***Ethernet*** correspondante. Cela est nécessaire, car lors de l'exécution de l'algorithme de conversion, le GF devra envoyer une requête à la classe d'émulation du convertisseur d'adresses. Il obtiendra en retour l'adresse de destination Ethernet correspondante au paquet ***IEEE1394*** traité. De plus, le contrôleur d'émulation devra générer une étiquette incluant la taille du paquet ***IEEE1394***, l'adresse où le paquet est placé dans la mémoire principale et un code correspondant au protocole ***IEEE1394***. Cette étiquette sera injectée dans le GF par l'intermédiaire de la classe d'émulation du contrôleur. Suite à ces stimulations, le GF exécutera l'algorithme chargé dans sa mémoire d'instructions sur le paquet ***IEEE1394***. En appliquant cet algorithme, le GF devrait répondre en plaçant l'en-tête ***Ethernet*** converti dans la mémoire temporaire et en générant des étiquettes destinées à l'assembleur de paquets. Toutes ces informations seront automatiquement recueillies par la classe d'émulation de la mémoire temporaire et par la classe d'émulation de l'assembleur de paquets, et elles seront transmises au contrôleur d'émulation du GF.

Pour les classes de vérification qui permettent de détecter les erreurs en cours de simulation, nous avons trois modules permettant d'effectuer cette tâche. Premièrement, il y a le vérificateur d'assertions. Le vérificateur d'assertions permet de s'assurer que certaines règles sont respectées en tout temps. Les règles à respecter sont directement extraites de la définition des cas de test dédiés au GF. Pour chaque règle, une assertion est définie à l'aide de l'algèbre temporelle fournie avec le langage ***e***.

Ensuite, la deuxième classe de vérification est le vérificateur d'exécution d'instructions du GF. Un problème posé par la vérification du GF est que, en plus d'avoir plusieurs interfaces avec son environnement externe, le traitement effectué par le GF est fortement encapsulé. Par exemple, lorsque le GF exécute une séquence de plusieurs instructions arithmétiques, tout le traitement s'effectue à l'intérieur du design sans que le banc d'essai ait d'informations sur la validité des opérations effectuées. Ainsi, il est fortement probable, dans le contexte d'un traitement encapsulé, qu'une erreur se produise et qu'elle

soit masquée par une autre opération effectuée sans que nous puissions détecter l'erreur sur ces interfaces externes. Afin de contrer ce problème, le module vérificateur d'exécution d'instructions du GF effectue une exécution externe (dans le banc d'essai) des instructions qui sont injectées dans le DUT. L'exécution externe des instructions injectées dans le DUT est réalisée en gardant une image des instructions placées dans le DUT. De plus, le vérificateur conserve son propre fichier de registres (16 entiers de 32 bits), son compteur de programme (8 bits) et un registre d'états (32bits).

La réalisation de ce vérificateur est très simple car le jeu d'instructions du GF a été modélisé à un haut niveau d'abstraction. Ainsi, la classe modélisant le jeu d'instructions ne contient pas seulement les champs de données de toutes les instructions, mais aussi une fonction de transfert pour chaque instruction. Cette fonction de transfert est l'effet que l'instruction produit sur le compteur de programme, le registre d'états et le fichier de registres. Ainsi, l'exécution d'une séquence d'instructions est effectuée en suivant les variations sur le compteur de programme du vérificateur d'instructions et en appelant les fonctions de transfert des instructions à exécuter. Ensuite, à un intervalle qui est configurable, le vérificateur effectue une comparaison entre son compteur de programme, son registre d'états et son fichier de registres avec ceux du GF.

Avec le vérificateur d'exécution d'instructions, il est possible de détecter les erreurs internes pouvant survenir dans l'exécution d'une instruction par le GF. Il y a cependant un cas particulier dans lequel le vérificateur fonctionnera dans un mode spécial. Ce cas particulier est lorsqu'une instruction accède aux interfaces externes du GF. Précisément, ce cas est activé par les instructions LD, LDI, ST, STI qui effectuent des chargements ou des rangements externes. Le vérificateur va observer l'interface du GF impliqué dans l'instruction afin d'obtenir la donnée dans le cas d'un chargement et vérifier si la donnée est bonne dans le cas d'un rangement. Cela est nécessaire afin que le vérificateur d'exécution puisse préserver son fichier de registres avec les mêmes ressources que celui du GF.

La dernière classe de vérification est le « scoreboard » qui est placé dans le contrôleur d'émulation. Ce module est utile lors de l'injection de paquet ***IEEE1394*** dans le DUT. Le « scoreboard » recevra une copie conforme du paquet ***IEEE1394*** injecté dans le DUT. Ensuite, lorsque le GF aura produit un résultat (un en-tête ***Ethernet***), il sera envoyé dans le « scoreboard ». Finalement, il s'occupera de vérifier si l'en-tête ***IEEE1394*** a bien été converti au format ***Ethernet***. Donc, le « scoreboard » comportera aussi une fonction de transfert permettant d'appliquer l'algorithme de conversion de protocoles dans le banc d'essai.

En guise de conclusion de ce chapitre, nous avons présenté les différentes fonctionnalités et interfaces reliées à notre processeur embarqué, le GF. De plus, nous avons discuté du jeu d'instructions du processeur. La vérification étant une étape cruciale dans la réalisation d'un modèle de processeur, nous avons donné une description de la méthodologie que nous avons adoptée, ainsi que les différents tests qui ont été appliqués, afin de valider notre processeur. Au cours du prochain chapitre, nous montrerons en détail les différentes réalisations pour l'implantation du processeur embarqué. Une comparaison entre notre processeur et un processeur embarqué commercialement disponible, en l'occurrence le ***ARM7***, sera aussi fournie afin de confirmer l'intérêt d'un processeur embarqué fait sur mesure.

CHAPITRE IV

Implantations et résultats

Ce dernier chapitre présente le travail qui a été fait dans le cadre de la réalisation du modèle du processeur embarqué GF. Il décrit les résultats du code VHDL ciblant trois différentes mises en oeuvre du processeur embarqué, notamment, la machine qui exécute une instruction par cycle d'horloge que nous nommerons ***GF_1***, la machine qui exécute une instruction par deux cycles que nous nommerons ***GF_2*** et la machine qui exécute une instruction par cycle avec un pipeline de deux instructions que nous nommerons ***GF_2p***.

Nous commencerons le présent chapitre en présentant l'implantation de l'assembleur qui a servi au développement de plusieurs applications, entre autres, celle de la conversion de protocoles ***Firewire*** à ***Ethernet***.

Ensuite, nous présenterons les différentes versions du processeur embarqué avec une comparaison de performance pour en retenir une version définitive. Nous montrerons les différents résultats relatifs à la technologie 0.35 µm à triple couche de métal.

Finalement, nous présenterons une comparaison entre notre version finale du processeur embarqué, ***GF_2p*** et un processeur embarqué existant sur le marché, le ***ARM7TDMI***.

4.1 Assembleur

Il s'agit de l'outil qui traduit le langage machine à un micro-code binaire. L'assembleur fournit une représentation plus familière qu'une simple représentation en '0' et '1'. Ceci a pour but de simplifier l'écriture et la lecture des programmes. Les noms symboliques des

opérations, ainsi que les locations sont une facette de cette représentation. Une autre facette est la facilité de programmation qui augmente la clarté du programme. Par exemple, l'utilisation des ***macros*** permet au programmeur d'étendre son langage machine en définissant de nouvelles opérations.

Le mandat de produire un assembleur pour le jeu d'instructions du ***GF*** a été confié à monsieur Armin Schneider, stagiaire de l'institut de recherche franco-allemand ISL en France. L'assembleur fourni par M. Schneider à la fin de son stage, est fonctionnel à 100% et a été vérifié à plusieurs reprises. Dans le cadre du présent projet, nous avons dû rajouter quelques nouvelles opérations pour enrichir notre jeu d'instructions. Le programme s'exécute sous deux systèmes d'exploitation, notamment, ***Linux*** et ***SUN-OS (Solaris)***. Il lit le fichier texte contenant le code de l'application en langage machine du GF et génère trois fichiers qui sont les suivants :

- Un fichier avec l'extension ***.lst*** qui est très utile pour la documentation. Ce fichier contient la version reformatée du code source avec le code binaire généré ainsi que les adresses en format hexadécimal;
- Un fichier binaire avec l'extension ***.bin*** qui contient le code binaire avec lequel nous pourrions programmer une puce programmable telle qu'un ***EEPROM*** (*Electrically Erasable Programmable Read-Only Memory*);
- Un fichier texte avec l'extension ***.out*** contenant le code binaire généré mais, dans un format lisible par l'usager.

Les figures 4-1et 4-2 illustrent un exemple d'un code d'assembleur et d'un code binaire respectivement.

```
entry :          LDIL   R10,0x0000      ; init R10 to 0
                 LDIH   R10,0x0000
                 LDIL   R11,0x0001      ; init R11 to 1
                 LIDH   R11,0x0000
                 LIDIL  R12,0xFFFF      ; init R12 to 0xFFFFFFFF
                 LIDH   R12,0xFFFF
                 RJMP   2               ; to main
interrupt:       XOR    R13,R13,R13     ; no interrupt defined yet
; 1-- read the headers
main:            LDIL   R0,0x0000       ; Address of Packet Ethernet
                 LD     R1,R0,0,3       ; Destination Address (LO,
MID)
                 LD     R2,R0,1,3       ; Dest (HI) + Source (LO)
address
                 LD     R3,R0,2,3       ; Source (MID, HI) address
                 LD     R4,R0,3,3       ; Data length (lo halfword)
                 LDB    R5,0,R0,2       ; RR5 = Source address LO
                 LDB    R5,0,R0,3
```

Figure 4.1 Un exemple de code d'assembleur

```
ADDR  DATA                                  lsb
0     00000000000000000001111100011010
1     00000000000000000001111100100010
2     00000000000000000011111101011010
3     00000000000000000001111110100010
4     00001111111111111111111110011010
5     00001111111111111111111110100010
6     00000000000000000000000010001011
7     00000000111111111111111111101000
8     00000000000000000000001010011010
9     01100000000000010100101000010010
a     01100000000010010100101010010010
b     01100000000010001010110010010010
c     01100000000110010100101110010010
```

Figure 4.2 Un exemple de code binaire

4.2 *Architecture interne du processeur embarqué*

Nous avons commencé notre étude par l'implantation de la machine ***GF_1*** i.e qui traite les instructions en un cycle d'horloge. Cette dernière utilise des modules de mémoire synthétisés directement avec l'outil de synthèse ***Synopsys*** à partir de la description ***RTL*** (*Register Transfer Language*). Ceci a produit une aire de dé très large. Nous avons analysé la complexité du circuit pour en sortir les points chauds. La mémoire

d'instructions de 256*32 prenait 87 % de la surface totale du circuit. Alors, nous avons décidé d'explorer l'avantage d'utiliser le compilateur de mémoire de la compagnie ***Virage Logic*** [43], qui est devenu disponible au cours de cette recherche. Les modules mémoires produits par le compilateur ***Virage*** sont synchrones et ne peuvent pas être utilisés pour réaliser la machine ***GF_1***. Cette dernière nécessite une mémoire d'instructions asynchrone qui ne peut pas être générée par le compilateur de mémoire. Nous avons donc été forcés de considérer la machine multicycle ***GF_2*** et la machine pipelinée ***GF_2p***. L'architecture interne du ***GF_2p*** est montrée à la figure 4.3. Elle consiste principalement en une collection des registres et une unité arithmétique et logique (ALU), reliée à un certain nombre de bus. Les registres accessibles au programmeur sont implantés en utilisant un fichier de registres à trois ports de 16 mots.

Les deux opérandes des ports 1 et 2, en l'occurrence, op1 et op2 respectivement, sont reliés aux entrées de l'ALU. Le résultat de l'opération est mis sur le bus de résultat. Ce dernier peut être écrit de nouveau dans le fichier temporaire des registres en utilisant le port 3. Le registre de compteur de programme (***PC***) fournit également l'opérande op1 pour les cas de branchements. Le registre d'états ***SREG*** est mis à jour à la suite de chaque opération. Une mémoire d'instruction de 256 mots de 32 bits est ajoutée dans le chemin des données.

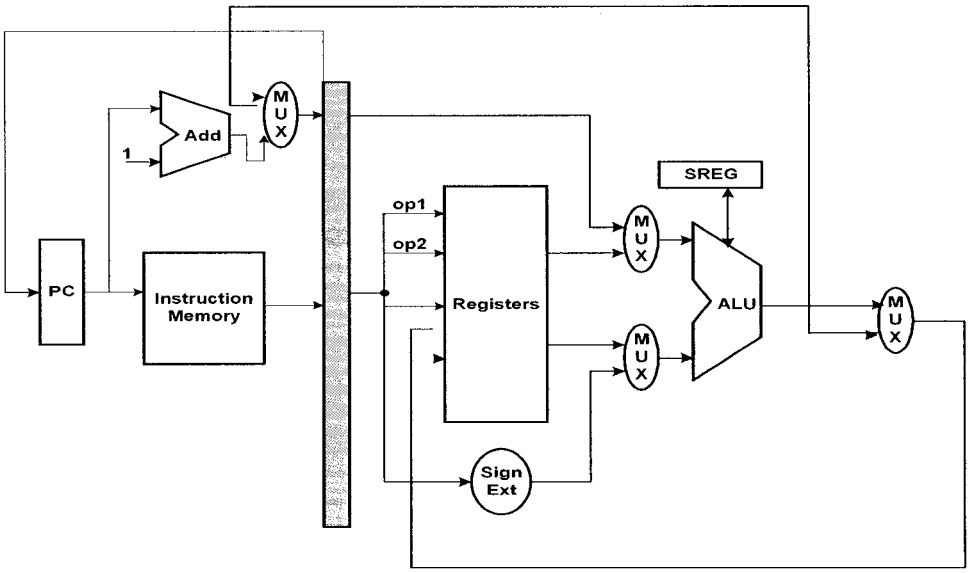


Figure 4.3 Schéma bloc de haut niveau de l'architecture du processeur embarqué

4.3 Résultats de synthèse sur un FPGA

Les résultats de l'implantation du processeur embarqué sur FPGA ont fourni une foule d'informations utiles par rapport aux problèmes de synthèse. Plusieurs simulations post-synthèse ont été faites en utilisant les outils de la plate-forme ARM-FPGA [5] disponibles dans les laboratoires du Groupe de Recherche en Microélectronique (GRM). Plusieurs défauts de conception ont été corrigés à ce stade. Indépendamment de la simulation post-synthèse usuelle en ***verilog***, l'implantation sur FPGA a donné une manière fiable pour vérifier les résultats de synthèse. La famille de Xilinx ***XCV1000*** [49] a été explorée pour l'usage dans un cas de prototypage.

La carte de prototype disponible dans le laboratoire du GRM contient un seul FPGA de type Xilinx-XCV1000-5. Il s'est avéré évident qu'un seul FPGA serait suffisant. La fréquence cible de 200 MHz pour la majeure partie de la conception est cependant quasiment impossible à rencontrer avec cette technologie, à moins que la fonctionnalité du module ne soit très limitée, ce qui n'est pas le cas. La fréquence de fonctionnement pour le processeur embarqué implanté sur FPGA est 30 MHz. L'incapacité d'atteindre la

fréquence cible est due principalement à la technologie du FPGA et à la longueur du chemin critique.

4.4 Résultats de synthèse sur un circuit intégré ASIC

Suite aux difficultés rencontrées avec l'implantation sur FPGA, nous avons porté notre attention sur la conception du GF sous une forme ASIC avec la technologie TSMC à 0,35 µm avec trois niveaux de métal. Malheureusement, la première version de notre processeur embarqué, GF_1, était très large (41 mm^2) en comparaison de seulement quelques mm^2 pour des processeurs comme le ARM7.

La dernière version du processeur embarqué ***GF_2p*** respectait une fréquence de 100 MHz après synthèse avec une marge confortable. L'outil de ***Cadence*** pour le placement et routage automatique était capable de maintenir cette fréquence cible. Les prochaines sections résumeront les performances pour les trois versions du processeur embarqué. La version pipelinée du processeur embarqué contenait 5,000 portes logiques sans tenir compte de la mémoire d'instructions qui a été générée par le compilateur de Virage Logic mais en incluant le fichier de registre.

4.5 Performance de notre processeur embarqué

Notre jeu d'instructions a été optimisé pour les manipulations de données exigées par l'application considérée, soit la conversion de protocoles. Une façon d'illustrer l'efficacité du jeu d'instructions est de considérer un exemple tel que déplacer le $2^{ième}$ octet d'un registre de 32 bits, R3, au $3^{ième}$ octet dans un autre registre de 32 bits, R7. En utilisant notre jeu d'instructions proposé, cette opération est exécutée avec une instruction, c.-à-d. ***LDB R7, 0x3, R3, 0x2*** comparée à 13 instructions dans le cas d'un processeur tel que le « Promic » [26]. Le tableau 4-1 résume les différentes instructions pour un transfert.

L'utilisation de notre processeur embarqué fait sur mesure, est très efficace, puisque nous pouvons indiquer notre propre registre de code condition et même ajouter de nouveaux codes pour des versions ultérieures. Ainsi, nous pourrions assigner un indicateur pour chaque interface reliée au processeur embarqué telle que le coprocesseur de vérification d'adresses, la mémoire principale, etc. Servir une interface exigerait l'utilisation des interruptions dans le cas d'un processeur ***ARM7***, qui prendrait énormément de temps en matière de coups d'horloge. En fait, avec notre processeur embarqué, l'interruption peut-être remplacée par une instruction de branchement qui prend un cycle d'horloge si la condition est satisfaite.

Tableau 4.1 Jeu d'instructions du GF contre celui d'un processeur RISC

GF	***Promic***
LDB R7, 0x3, R3, 0x2	TFR R4, R3 // move R3 in R4 XOR R5, R5 // content of R5 is null ADI R5, 0x00FF0000 // content of R5 is the mask AND R4, R5 // masking operation LSR R4 // logical shift to the right LSR R4 LSR R4 LSR R4 LSR R4 LSR R4 LSR R4 LSR R4 OR R7, R4 // write

Une autre différence significative qui milite en faveur de notre processeur embarqué est au niveau des instructions de chargement et de rangement qui prennent 3 et 2 coups d'horloge respectivement dans le cas du processeur ***ARM7***. Ces deux opérations prennent seulement 2 et 1 coups d'horloge respectivement dans le cas de notre processeur embarqué. Des opérations de transfert sont fréquemment utilisées dans des applications de transfert de données telles que la conversion de protocoles et la classification de paquets.

Les utilisateurs et les concepteurs évaluent souvent la performance des processeurs avec différentes métriques. Si nous pouvions trouver une relation qui tient compte de toutes ces métriques, nous pourrions déterminer l'effet d'un changement de conception sur la performance vue par l'utilisateur. Puisque nous nous limitons à l'exécution de l'unité centrale de traitement en ce moment, le point important est le temps d'exécution de l'unité centrale de traitement. Ainsi, nous avons utilisé l'équation de base de la performance donnée par [32] pour évaluer notre processeur embarqué et le comparer avec le processeur ARM7 :

$$\text{Temps d'exécution} = \frac{\text{Instructions}}{\text{Programme}} \text{x} \frac{\text{Cycles\_horloge}}{\text{Instruction}} \text{x} \frac{\text{Secondes}}{\text{Cycles\_horloge}}.$$

Cette formule est particulièrement utile parce qu'elle sépare les trois facteurs principaux qui affectent la performance, à savoir le compte d'instruction (le nombre d'instructions exécutées par le programme), le nombre de cycles par instruction et la période d'horloge.

Comme mentionné précédemment, nous avons mis en œuvre trois versions de notre processeur embarqué, à savoir la machine simple qui opère en un cycle (GF_1), la machine multicycle qui utilise deux cycles par instruction (GF_2) et la machine pipelinée en deux étages (GF_2p). Le tableau 4.2 récapitule les différents paramètres de performance post-synthèse en utilisant l'outil de ***Synopsys*** [version 2001.08-sp2]. Il est à noter que le placement et routage a été fait à l'aide des outils de ***cadence*** [version 5.3] en utilisant la technologie 0.35 µm de TSMC avec trois niveaux de métal.

Tableau 4.2 Performances des trois versions du processeur embarqué

	GF_1	GF_2	GF_2p
Vdd (Volt)	3.3	3.3	3.3
Fréquence (pire cas)	90 MHz	145 MHz	100 MHz
Estimation de la surface du dé	9.4 mm^2	1.2 mm^2	1.3 mm^2
Surface actuelle du dé	41 mm^2	n.d.	5.3 mm^2
Technologie	0.35 µm	0.35 µm	0.35 µm
Cache	256-Dword	256-Dword	256-Dword
Coprocesseur mathématique intégré	Aucun	Aucun	Aucun
ISA	RISC	RISC	RISC
Longueur d'instruction	32 bits	32 bits	32 bits
Puissance dissipée	14.74 mW/MHz	1.14 mW/MHz	3.37 mW/MHz

Le tableau 4.3 compare la dernière version du processeur embarqué soit le ***GF_2p*** avec le processeur ***ARM7***. Il est à noter que la surface totale du dé du processeur ***ARM7*** n'inclut pas la mémoire locale d'instructions. Par conséquent, nous devrions ajouter une surface additionnelle de 0,1 mm^2 pour une mémoire d'instructions de 256 mots de 32 bits. L'outil de conception utilisé pour générer une telle mémoire est le compilateur de mémoire de la compagnie Virage Logic [43]. Il a été, également, utilisé dans les deux versions ***GF_2*** et ***GF_2p*** pour générer la mémoire d'instructions.

Tableau 4.3 Performances de la version définitive du processeur embarqué contre *ARM7TDMI*

	ARM7TDMI	GF_2p
Vdd (Volt)	3.3	3.3
Fréquence (pire cas)	45 MHz	100 MHz
Surface actuelle du dé	2.14 mm^2	5.3 mm^2
Technologie	0.35 µm	0.35 µm
Cache	Aucun	256-Dword
Puissance dissipée	1.90 mW/MHz	3.37 mW/MHz

Considérant que notre processeur embarqué est plus simple qu'un processeur ***ARM7***, il est surprenant qu'il occupe une surface de dé qui est le double de la taille d'un processeur ***ARM7***, avec la même technologie de 0,35µm-CMOS de la compagnie TSMC. Il est difficile d'expliquer cette différence, puisque nous n'avons pas accès aux informations sur la disposition du processeur ***ARM7***, au-delà de l'information du domaine public trouvée sur leur site Web [6]. Cependant, nous pouvons attribuer la taille relativement large de notre processeur comparée à celle du processeur ***ARM7***, par l'utilisation des concepteurs de la compagnie ARM d'une méthodologie de conception physique optimisée. En outre, les caractéristiques de la bibliothèque de cellules fournie par TSMC pourraient avoir un impact fort, dépendant si les cellules ont été conçues pour la fréquence, la densité ou pour une basse consommation de puissance. Nous n'avons aucun contrôle sur la disposition physique de ces cellules. Elles ont été mises à notre disposition sous la forme de boîtes noires fournies par la compagnie TSMC et via la société canadienne de la microélectronique CMC [41]. Ces cellules sont automatiquement placées et routées par l'outil de placement et routage de ***Cadence*** en utilisant un processus de trois niveaux de métal. Pour conclure sur ce point, nous considérons que la différence de la surface de dé que nous avons trouvée est une conséquence de notre utilisation d'un noyau doux « soft-core », où la disposition est faite par l'utilisateur. Par contre, l'utilisation d'un noyau dur « hard-core », où le fournisseur fait lui-même la disposition pour une performance optimale, peut donner des performances bien supérieures.

De nos jours, il est à noter que le meilleur type de programmes à utiliser pour des fins de calcul de performance sont des vraies applications. Celles-ci peuvent être des applications que l'usager utilise régulièrement ou simplement des applications typiques. Dans notre cas, nous avons choisi la conversion du protocole ***Firewire*** en ***Ethernet***. Cette application a été codée en utilisant les langages d'assembleur des processeurs considérés pour éliminer les effets du compilateur. En utilisant notre définition de temps d'exécution déjà

donnée, nous avons obtenu les résultats suivants pour les trois versions de notre processeur embarqué :

Temps d'exécution $_{ARM7}$ = 125 cycles @ 45 MHz = 2778 ns
Temps d'exécution $_{GF\_1}$ = 71 cycles @ 90 MHz = 789 ns
Temps d'exécution $_{GF\_2}$ = 142 cycles @ 145 MHz = 979 ns
Temps d'exécution $_{GF\_2p}$ = 75 cycles @ 100 MHz = 750 ns

L'approche la plus simple pour calculer la performance relative est d'utiliser le temps d'exécution total pour chaque cas. Ainsi,

$$\frac{\text{Performance (GF\_1)}}{\text{Performance (ARM7)}} = \frac{\text{Temps\_Exec.(ARM7)}}{\text{Temps\_Exec.(GF\_1)}} = 3.5$$

$$\frac{\text{Performance (GF\_2)}}{\text{Performance (ARM7)}} = \frac{\text{Temps\_Exec.(ARM7)}}{\text{Temps\_Exec.(GF\_2)}} = 2.8$$

$$\frac{\text{Performance (GF\_2p)}}{\text{Performance (ARM7)}} = \frac{\text{Temps\_Exec.(ARM7)}}{\text{Temps\_Exec.(GF\_2p)}} = 3.7$$

4.6 Discussions

Dans cette section, nous discuterons de quelques options de conception afin d'attirer l'attention du lecteur sur l'existence de celles-ci.

4.6.1 Les processeurs configurables

Une des compagnies majeures dans le domaine des modèles de processeurs configurables est Tensilica. Celle-ci est un fournisseur de blocs de propriété intellectuelle, principalement des processeurs de type RISC. L'architecture du processeur Xtensa, le principal produit de Tensilica, est une architecture de type 32 bits qui pourrait

correspondre aux besoins des systèmes embarqués. Les caractéristiques de base de ce processeur sont les suivantes [42] :

- Complexité d'environ 25 000 portes logiques, ce qui donne une utilisation d'environ 0.7 mm^2 de surface de silicium, pour une implantation en technologie 0.18µm;
- Haute performance : plus de 220 MIPS pour une fréquence d'horloge de 200 MHz;
- Faible consommation de puissance : 0.4 mW/MHz pour une configuration typique, avec une implantation en technologie 0.18 µm.

4.6.2 Le processeur logiciel MicroBlaze offert par la compagnie Xilinx

Le processeur logiciel ***MicroBlaze*** de Xilinx [50] roule à 125 MHz avec un bus de 32 bits d'instructions et de données. C'est un noyau qui est considéré rapide. En fait, il peut être implanté comme processeur autonome dans un FPGA de type ***Spartan II*** ou ***Virtex***. Mais, puisqu'il consomme seulement 800 LUTs (*Look-Up Table*), le noyau peut également être mélangé et assorti avec les processeurs et les périphériques additionnels de MicroBlaze sur des dispositifs de Virtex pour créer une plate-forme de multiprocesseur.

Le ***MicroBlaze*** est un processeur de 32 bits, basé sur un processeur RISC standard, avec un fichier temporaire de registres de 32 mots de 32 bits. Ce dernier peut être implanté en utilisant le BlockRAM du FPGA et/ou une mémoire externe. Un des avantages de cette technologie est le suivant : elle permet à l'utilisateur d'accorder une taille désirée à la mémoire cache, selon ses besoins pour une application ou un programme donné. Le choix de la mémoire peut être parmi le suivant : 2 KO, 4 KO, 8 KO, 16 KO. La figure 4.3 illustre l'architecture interne du processeur ***MicroBlaze*** offert par la compagnie Xilinx.

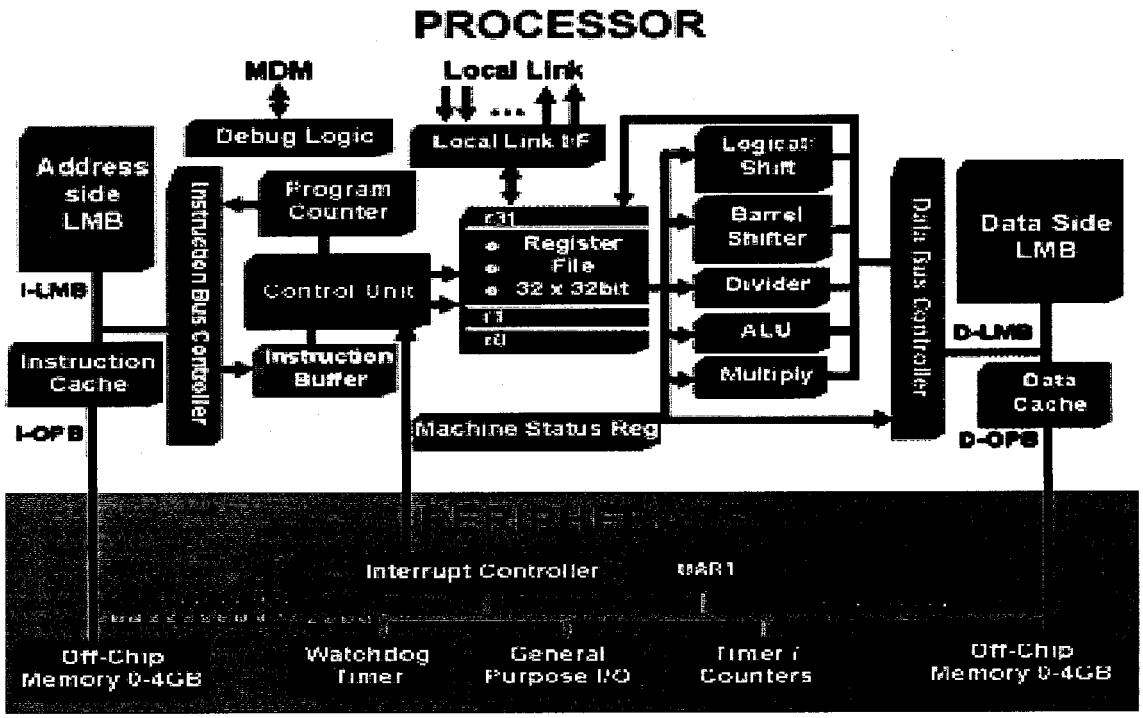


Figure 4.3 Architecture interne du processor logiciel *MicroBlaze* de Xilinx [50]

En guise de conclusion, un processeur embarqué dédié à une plate-forme SoC pour le traitement de réseau a été présenté. Il est conçu sur mesure pour augmenter la flexibilité. Nous avons montré que le processeur embarqué proposé est 3,7 fois plus rapides qu'un processeur embarqué existant tel que le processeur *ARM7*. Ce mémoire démontre quelques avantages d'un processeur embarqué dans une plate-forme SoC consacrée aux paquets de la vidéo numérique. Il est programmable, flexible, et facile à développer. Un processeur embarqué conçu sur commande est une alternative viable à utiliser versus un module « IP » *ARM7* très coûteux. Il offre un rendement plus élevé, et peut-être facilement adapté à nos besoins; cependant, nous n'avons pas pu pas approcher la surface de dé d'un processeur *ARM7* avec la bibliothèque de cellules disponible et la méthodologie de conception physique.

CONCLUSION

Le point de départ pour réaliser le modèle exécutable d'une plate-forme SoC dédié pour les processeurs réseau a été la définition des spécifications. Cette première étape, qui a duré plusieurs mois, conditionne naturellement le succès futur du projet. Il s'agit, en effet, de pousser l'exploration et autres recherches bibliographiques afin de proposer une première architecture simplifiée pour le modèle exécutable, sans pour autant compromettre d'éventuelles évolutions et le design physique.

La phase de recherche s'est donc essentiellement concentrée sur les nombreuses notions liées aux protocoles. Celles-ci se révèlent être le véritable cœur du projet. Un aperçu sur les solutions existantes a permis aussi de cerner les limites des produits présents sur le marché et de voir dans quel sens devait tendre l'architecture : le paramètre retenu a été la flexibilité. Notre plate-forme doit être capable de convertir divers protocoles, selon le programme d'exécution qu'il a chargé en mémoire.

Le développement du processeur embarqué ***GF*** a été inspiré de modèles de processeurs RISC 32 bits, spécialisés dans la manipulation de données. Cependant, le jeu d'instructions est réellement spécifique aux besoins de la conversion, définis lors de l'étude des algorithmes de conversion des protocoles.

Notre architecture se devait non seulement d'être flexible, mais aussi de réaliser la conversion en diminuant la latence à son minimum. Dans ce but, l'exécution des instructions doit prendre un minimum de cycles d'exécution. Les résultats obtenus lors des tests de cette description de type comportementale sont conformes à ces exigences.

Le premier chapitre constitue une revue des travaux qui ont été réalisés dans le domaine des processeurs réseau. En effet, nous avons fait un survol de l'évolution des

commutateurs à haut débit ainsi que plusieurs processeurs réseau qui sont déjà commercialisés. Une architecture réussie de plate-forme SoC dédiée pour des processeurs réseau doit prendre en considération non seulement la performance exigée de nos jours, mais aussi l'amélioration nécessaire pour des fonctionnalités futures. Chaque architecture proposée doit être minutieusement examinée pour associer horloge, vitesse, complexité et flexibilité de programmation. Nous avons pu voir dans le cadre de ce premier chapitre la puissance des processeurs réseau en raison de leur programmabilité et de la spécialisation de ces processeurs dans les applications de télécommunication. Dans l'avenir, ils prendront certainement la place des processeurs RISC et des circuits intégrés ASIC dans les commutateurs de grande capacité. Également, nous avons discuté des avantages de performance des processeurs réseau sur les processeurs RISC qui étaient d'utilisation générale et de leurs nombreux avantages par rapport aux circuits intégrés ASIC.

Dans le deuxième chapitre, nous avons présenté une description la plus exhaustive possible de l'architecture d'une plate-forme SoC dédiée à des processeurs réseau mais, particulièrement à la conversion des protocoles. Elle a été conçue pour réaliser de façon fonctionnelle la conversion d'un paquet supporté par le protocole Ethernet en un paquet supporté par le protocole Firewire. En effet, c'est cette première version exécutable qui servira de base de travail pour développer une plate-forme SoC performante, flexible et capable de traiter le plus grand nombre de protocoles des différentes couches du modèle OSI. L'architecture présentée dans ce chapitre est donc le point de départ d'un vaste projet visant l'obtention d'une plate-forme SoC générique, dédiée à des processeurs réseau. Elle sera éventuellement implantée au final dans un ASIC. Du fait de la complexité croissante des nouveaux protocoles et de la très faible normalisation qui existe dans ce domaine, il est certain que ce type d'architecture va devenir de plus en plus indispensable pour permettre aux équipements de générations différentes de communiquer. Ceci est d'autant plus vrai qu'avec l'incroyable développement des services de l'Internet, l'importance de la transmission par réseaux de données vidéo augmente chaque jour de façon irréversible.

Le troisième chapitre a traité du modèle de processeur embarqué qui agira comme cœur de notre plate-forme SoC dédiée à des processeurs réseau. Ce processeur embarqué emmagasine localement le code du programme nécessaire aux applications de télécommunication, notamment la conversion de protocoles. Le jeu d'instructions a été basé sur celui d'un processeur DLX mais, nous avions soigneusement ajouté plusieurs nouvelles instructions qui ont pour but d'accélérer le traitement de paquets de type vidéo numérique à très haut débit. Ensuite, nous avons présenté les différentes interfaces qui gèrent la communication entre notre processeur embarqué et les divers coprocesseurs (accélérateurs matériels) au sein de notre plate-forme. Finalement, nous avons décrit le plan de vérification nécessaire pour assurer que ce processeur soit de qualité suffisante. La méthodologie de vérification utilise les concepts orientés objet et orientés aspect avec comme outil Specman Elite™.

Le quatrième et dernier chapitre a présenté le travail qui a été réalisé dans le cadre de l'implantation et le calcul de performance du processeur embarqué. Nous avons commencé par décrire brièvement l'assembleur qui a été développé et que nous avons utilisé pour monter une application de conversion de protocoles, en l'occurrence, la conversion du protocole Firewire au protocole Ethernet. Cette application a servi pour calculer la performance de notre processeur et celle d'un processeur ***ARM7***. Ce dernier a été utilisé comme point de référence. Puis, nous avons donné les résultats de synthèse de notre processeur pour un circuit FPGA et pour un circuit intégré ASIC avec une technologie 0,35 µm. Finalement, nous avons terminé ce chapitre en présentant la nécessité et les avantages de notre processeur embarqué et de son jeu d'instructions. Une comparaison entre notre processeur et le processeur ARM7 a été faite pour démontrer ces avantages. Nous avons obtenu une performance de 3,7 fois mieux par rapport au processeur ARM7 basé sur notre application de conversion de protocoles.

Toutefois, il serait maintenant, intéressant de développer une comparaison basée sur l'application de conversion de protocoles et possiblement d'autres applications,

notamment la classification de paquets avec le jeu d'instructions d'un processeur réseau tel que Intel IXP1200. Nous avons déjà reçu une plate-forme de la compagnie Intel qui nous permettra d'effectuer une telle comparaison. Par ailleurs, il faut également adapter la dernière version du processeur embarqué avec la dernière version de l'architecture de la plate-forme.

En guise de conclusion, ce mémoire a montré le besoin et les avantages d'un processeur embarqué au sein d'une plate-forme SoC dédiée à des processeurs réseau. Nous avons montré que la conception d'un processeur embarqué fait sur mesure, pouvait donner d'excellents résultats, comparé au processeur ***ARM7*** disponible sur le marché. Toutefois, nous n'avons pas obtenu de bons résultats concernant la conception physique de notre module à cause de la méthodologie utilisée et des cellules de la bibliothèque disponible.

RÉFÉRENCES

[1] AGERE INC., "Building Next Generation Network Processors", White Paper, 1999.

[2] AGERE INC., "The Challenge for Next Generation Network Processors", White Paper, 1999.

[3] ALCHEMY SEMICONDUCTOR INC., "The Alchemy Au1000 Internet Edge Processor", Product brief, 2000.

[4] ALEXANDER P., GELINAS R., HAYS W. P., KATZMAN S., DALLY W. J., "NetVortex: A Scalable, Multiprocessor for Network Communications", Presentation. Embedded Processor Forum, 14 juin, 2000.

[5] ARM, "Integrator ASIC Platform development Boards (AP)'', 2001.

[6] ARM, http://www.arm.com/aboutarm/4X9FBU/$File/foundry_prog.pdf

[7] BERGERON J., "Writing testbenches, Functional verification of HDL models", Kluwer Academic Publishers, 2000.

[8] BRECIS COMMUNICATIONS, "MSP5000 Multi-Service Processor", Product Brief, mai 2001.

[9] CIAC, http://www.ciac.org/ciac/bulletins/m-050.shtml

[10] CISCO, http://www.cisco.com/en/US/products/hw/routers/ps233/prod_literature.html

[11] CLEARSPEED, http://www.clearspeed.com/index.php

[12] COLE B., "Intel net processor boosts clock, adds C compiler", EE Times, 20 février 2001.

[13] EZCHIP TECHNOLOGIES, "7-Layer Packet Processing: A Performance Analysis", White paper, juillet 2000.

[14] EZCHIP TECHNOLOGIES, "Network Processor Designs for Next-Generation Networking Equipment", White paper, décembre 1999.

[15] GWENAP L., "Net processor makers race toward 10-Gbit/s goal", EE Times, 19 juin 2000.

[16] GWENAP L., "Lexra offers NetVortex net processor as licensable core", EE Times, 12 juin 2000.

[17] HAQUE F.I.,. KHAN K. A., MICHELSON J., "The Art of Verification with VERA", Verification Central, 2001.

[18] HENNESSY J., WOLF W., "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, 2ième édition, p 645, 1996.

[19] HUSAK D., "Communication Processors: a definition and comparison", White paper Motorola Corp., 3 mai 2000.

[20] IBM CORP., "IBM Network Processor (IBM32NPR161EPXCAC100)", Product Overview, November 1999.

[21] IBM CORP., http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/7874C7DA8607C0B287256BF3006FBE54

[22] IEEE Computer Society, "IEEE 802.3".

[23] IEEE Computer Society, "IEEE 1394".

[24] INTEL, ftp://download.intel.com/design/network/datashts/27829810.pdf, IXP1200 Network Processor Datasheet décembre 2001.

[25] LEPAGE, R., "Méthode Co-design (Logiciel/matériel) d'identification et d'auto classification des protocoles de haut niveau", Mémoire de Maîtrise, Université du Québec à Montréal, décembre 2002.

[26] LIMAM J. K., VERNEL P., COLNET D., "Design of a customizable processor IP", International Workshop on IP Based Synthesis and System Design, p. 181-185, Grenoble, 1999.

[27] LUCENT TECHNOLOGIES, "PayloadPlus Fast Pattern Processor", Preliminary Product Brief. Lucent Technologies, Microelectronics Group, avril 2000.

[28] MAKER, "MXT4400: Traffic Stream Processor", Product Brief, 1999.

[29] MOTOROLA CORP., "C-5 Digital Communications Processor", Product brief, 4 mai 2000.

[30] MOTOROLA CORP., "C-Ware Software Toolset", Product Brief. 2 mai 2000.

[31] MMC NETWORKS, "nP Family", Data Sheet.

[32] PATTERSON D., HENNESSY J., "Computer Organization & Design", Morgan Kaufmann, 1998.

[33] QUANTUM EFFECT DEVICES, "QED RISCMark", Product Sheet.

[34] REGIMBAL S., LEMIRE J. F., SAVARIA Y., BOIS G., ABOULHAMID E.M., BARON A., "Aspect partitioning for Hardware Verification Reuse", IWSOC2002, 2002.

[35] ROY R., "A Monolithic Packet Processing Architecture", Presentation Network Processor Forum, June 14, 2001.

[36] SANKAR N., "CNP810^{TM} Network Services Processor Family", Presentation Network Processor Forum, 14 juin 2001.

[37] SIBYTE, "SB-1250", Data Sheet, octobre 2000.

[38] SIBYTE, "SB-1 CPU", Fact Sheet. octobre 2000.

[39] SITERA CORP., "PRISM IQ2000", Product Brief, février 2000.

[40] SJOHOLM S., LINDH L., "VHDL for Designers", Prentice Hall Europe, p 367, 1999.

[41] SOCIÉTÉ CANADIENNE DE MICROÉLÉCTRONIQUE, http://www.cmc.ca

[42] TENSILICA INC., Application Specific Microprocessor Solutions Overview Handbook.

[43] VIRAGE LOGIC INC., http://www.viragelogic.com./

[44] WASHINGTON UNIVERSITY, http://www.washington.edu/hdtv/sc99/reference.html

[45] WIKIPEDIA, http://www.wikipedia.org/wiki/Harvard_architecture

[46] WIRBEL L., "Network processor handles mix of carrier services", EE Times, 18 spetembre 2000.

[47] XELERATED PACKET DEVICES, "Xelerator™ T40 Traffic Manager", Preliminary Product Brief, juin 2001.

[48] XELERATED PACKET DEVICES, "Xelerator™ X40 Packet Processor", Preliminary Product Brief, juin 2001.

[49] XILINX, http://direct.xilinx.com/bvdocs/publications/ds003.pdf

[50] XILINX, http://www.xilinx.com/ipcenter/processor_central/microblaze/literature.htm

ANNEXE A.1

Détails des opérations du jeu d'instructions

ADD (ADDition without carry)

Syntax : ADD Ri, Rj, Rk

Definition : Addition of the contents of both registers Ri and Rj without the carry C (Ri + Rj).
The result is placed in register Rk.

Operation : S = A + B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	A31.B31+A31./S31+B31./S31
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	A31.B31./S31+/A31./B31.S31
IDLE	0

ADI (ADdition with an Immediate value)

Syntax : ADI Ri, #n

Definition : Addition of the contents of registers Ri with an immediate value coded on 16 bits.
The result is placed in register Ri.

Operation : S = A + B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	A31.B31+A31./S31+B31./S31
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕ S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./ S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	A31.B31./S31+/A31./B31.S31
IDLE	0

AND (logical AND)

Syntax : AND Ri, Rj, Rk

Definition : Logical "AND" between the contents of both registers Ri and Rj. The result is placed in register Rk.

Operation : S = A.B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	0
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕ S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./ S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	0
IDLE	0

IDLE (IDLE mode)

Syntax : IDLE

Definition : No instruction is performed during one clock cycle and the General Formatter is in IDLE mode (used for tests only).

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	1

LD (LoaD indirect)

Syntax : LD Ri, (Rj), #o, #s

Definition : Loads indirectly the contents of the memory module referred by #s at the address calculated with (Rj + #o).
The loaded data is placed in register Ri.
This instruction is used to read the Controller's tag, to send the address to be converted to the Addresses Checker/Converter and to get the converted address. If this instruction is used to load information from both memories, it must be followed by a NOP instruction

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

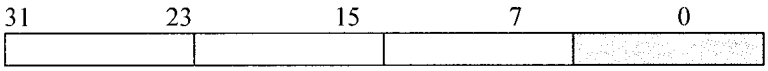
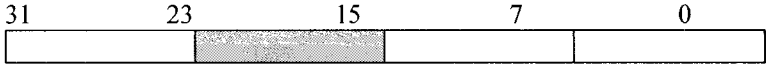
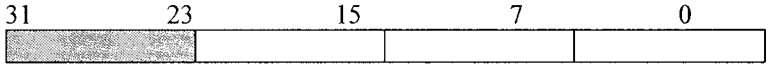
Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

LDB (LoaD Bytes)

Syntax : LDB Ri, #n, Rj, #m

Definition : Loads the byte "m" of the register Rj to the byte "n" of the register Ri.

m or n	Pointed byte of the register
00	
01	
10	
11	

The loaded data is placed in register Ri.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

LDI (LoaD with an Immediate value)

Syntax : LDI Ri, #@, #s

Definition : Loads the contents of the memory module referred by #s at the address #@.
The loaded data is placed in register Ri.
If this instruction is used to load information from both memories, it must be followed by a NOP instruction

Program Counter : PC + 1 → PC

Execution Time : 1 cycle
Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

LDIH (LoaD Immediate to High 2-bytes)

Syntax : LDIH Ri, #n

Definition : Loads an immediate 16-bit value into the bits 31 down to 16 of register Ri.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

LDIL (LoaD Immediate to Low 2-bytes)

Syntax : LDIL Ri, #n

Definition : Loads an immediate 16-bit value into the bits 15 down to 0 of register Ri.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

LSL (Logical Shift Left of n bits)

Syntax : LSL Ri, #n

Definition : Performs a logical shift of n bits to the left on register Ri. Bits (n-1) to 0 are cleared.
Ri[k] → Ri[k + n] for 31 ≥ k; n ≥ 0.
This instruction multiplies an unsigned value by (n+1).

Operation: S = A << n

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	S31
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	A31⊕S31
IDLE	0

LSR (Logical Shift Right of n bits)

Syntax : LSR Ri, #n

Definition : Performs a logical shift of n bits to the right on register Ri. Bits 31 to (31–n+1) are cleared.
Ri[k + n] → Ri[k] for 31 ≥ k; n ≥ 0.
This instruction divides an unsigned value by (n+1).

Operation: S = A >> n

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	S0
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	0
IDLE	0

MOV (MOVe between registers)

Syntax : MOV Ri, Rj

Definition : Transfers the content of register Rj into register Ri.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
OV	-
IDLE	0

MSK (MaSK between registers)

Syntax : MSK Ri, Rj, Rk

Definition : Performs the logical AND-NOT between the contents of the registers Ri and Rj.
The result is placed in register Rk.

Operation: S = A./B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	0
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	0
IDLE	0

NOP (No OPeration)

Syntax : NOP

Definition : No instruction is performed during one clock cycle.
No modification of the status register.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

OR (logical OR)

Syntax : OR Ri, Rj, Rk

Definition : Performs the logical OR between the contents of the registers Ri and Rj.
The result is placed in register Rk.

Operation: S = A+B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	0
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11./S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	0
IDLE	0

RB0 (Relative Branch if "0")

Syntax : RB0 #d, #n

Definition : Conditional relative branch. Tests the bit "n" of the status register and branches relatively to PC if the bit is cleared.
The offset is coded on 8 bits.

Program Counter : PC + d → PC (else PC + 1 → PC)

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

RB1 (Relative Branch if "1")

Syntax : RB1 #d, #n

Definition : Conditional relative branch. Tests the bit "n" of the status register and branches relatively to PC if the bit is set.
The offset is coded on 8 bits.

Program Counter : PC + d → PC (else PC + 1 → PC)

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

RJMP (Relative JuMP)

Syntax : RJMP #d

Definition : Jumps to a relative address within 0.25kwords (offset coded on 8 bits).

Program Counter : PC + d → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

RSTIDLE (ReSeT IDLE)

Syntax : RSTIDLE

Definition : No instruction is performed during one clock cycle and the General Formatter is in IDLE mode.
The Program Counter is not incremented.

Program Counter : PC → PC +1

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

ST (STore indirect)

Syntax : ST Ri, (Rj), #o, #s

Definition : Stores the content of the register Ri in the memory module referred by #s at the address calculated with (Rj + #o).

Program Counter : PC + 1 → PC

Execution Time : 1 cycle.

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

STI (STore with an Immediate)

Syntax : STI Ri, #@, #s

Definition : Stores the content of the register Ri in the memory module referred by #s at the address #@.

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	-
P	-
Z	-
N	-
V	-
IDLE	0

SUB (SUBstraction)

Syntax : SUB Ri, Rj, Rk

Definition : Subtracts the contents of the registers Ri and Rj and places the result in Rk.

Operation: S = A-B

Program Counter : PC + 1 → PC

Execution Time : 1 cycle

Status Register :

C	A31.B31+A31./S31+B31./S31
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13 ⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11. /S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	/A31.B31.S31+A31./B31./S31
IDLE	0

XOR (logical XOR)

Syntax : XOR Ri, Rj, Rk

Definition : Performs the logical XOR between the contents of the registers Ri and Rj.
The result is placed in register Rk.

Operation: $S = A \oplus B$

Program Counter: $PC + 1 \rightarrow PC$

Execution Time : 1 cycle

Status Register :

C	0
P	S31⊕S30⊕S29⊕S28⊕S27⊕S26⊕S25⊕S24⊕S23⊕S22⊕S21⊕S20⊕S19⊕S18⊕S17⊕S16⊕S15⊕S14⊕S13 ⊕S12⊕S11⊕S10⊕S9⊕S8⊕S7⊕S6⊕S5⊕S4⊕S3⊕S2⊕S1⊕S0
Z	/S31./S30./S29./S28./S27./S26./S25./S24./S23./S22./S21./S20./S19./S18./S17./S16./S15./S14./S13./S12./S11. /S10./S9./S8./S7./S6./S5./S4./S3./S2./S1./S0
N	S31
V	0
IDLE	0