Titre: Title:	Paramétrisation et reconfiguration automatique du chemin de données d'un processeur synthétisable
Auteur: Author:	Alexandre Fortin
Date:	2000
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Fortin, A. (2000). Paramétrisation et reconfiguration automatique du chemin de données d'un processeur synthétisable [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7814/

UNIVERSITÉ DE MONTRÉAL

PARAMÉTRISATION ET RECONFIGURATION AUTOMATIQUE DU CHEMIN DE DONNÉES D'UN PROCESSEUR SYNTHÉTISABLE

ALEXANDRE FORTIN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
AOÛT 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PARAMÉTRISATION ET RECONFIGURATION
AUTOMATIQUE DU CHEMIN DE DONNÉES
D'UN PROCESSEUR SYNTHÉTISABLE

Présenté par : FORTIN Alexandre

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

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

M. BOIS Guy, Ph.D., président

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

M. SAWAN Mohamad, Ph.D., membre et codirecteur de recherche

M. ABOULHAMID Moustapha, Ph.D., membre

RÉSUMÉ

Les processeurs de traitement de signaux (*DSP*) sont utilisés dans un nombre d'applications toujours grandissant. De plus, ceux-ci doivent être performants et optimisés aux tâches à accomplir tout en minimisant le temps de développement. Ce mémoire présente donc l'architecture d'un chemin de données hautement parallèle et configurable, de type SIMD (*Single Instruction Multiple Data*).

Le point de départ étant un processeur développé en vue d'une implantation dans un ASIC, la première partie du travail consistait à adapter l'architecture afin de porter l'implantation vers un FPGA de la série Virtex de Xilinx. La seconde étape de développement, plus complexe, nous a amené à effectuer plusieurs transformations du code VHDL, de telle sorte que le chemin de données puisse être complètement configurable. En effet, selon l'application demandée, l'architecture peut dorénavant être optimisée, lors de la synthèse du circuit, à l'aide de quelques paramètres contrôlant entre autres la largeur des mots de donnée, les dimensions des registres et des mémoires et le nombre d'éléments de calcul (*PE*) composant le chemin de données. De plus, le chemin de données est modulaire. En effet, chaque module composant un PE peut être individuellement inclus ou exclus de l'architecture, sans affecter le fonctionnement des autres modules. Toutes les stratégies déployées pour l'atteinte de ces résultats sont expliquées dans le présent mémoire, en accordant une importance particulière au multiplicateur en raison de son architecture difficilement paramétrable.

ABSTRACT

DSPs are used in a continuously growing number of applications. They should provide a high perfomance, while offering a short development time. This master thesis presents the architecture of a highly parallel and configurable datapath, which is a SIMD (*Single Instruction Multiple Data*) type.

The starting point of this work was a processor first developed as an ASIC. Hence, the first task to realize was to adapt the architecture to a FPGA implementation, specifically a model part of the Virtex family produced by Xilinx. The second stage of the development, more complex, was to modify the VHDL code of the datapath in order to make it completely configurable. In fact, the architecture can now be optimised to the desired application with only a few parameters that adapt the circuit during the synthesis phase. For example, the bit width of the data words, the depth of the registers and the memories, and the number of processing elements (*PE*) that compose the datapath are all parameterizable. In addition, the datapath is modular by allowing each module part of a PE to be included or excluded, without affecting the function of the other modules. All the strategies deployed to reach these results are explained in this thesis. Finally, a particular attention is given to the multiplier since its architecture is particularly complex to parameterize.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

ALU *(Arithmetic and Logic Unit)* Unité de calcul arithmétique et logique comprise dans un DSP.

ASIC *(Application-Specific Integrated Circuit)* Circuit Intégré conçu pour une application visée. Ils sont très performants et offrent le plus haut niveau d'intégration de tous les types de circuits intégrés.

CLB *(Configurable Logic Block)* Bloc de logique configurable constituant l'élément de circuit fondamental à l'intérieur d'un FPGA. Un assemblage de CLB liés forme un circuit logique.

DLL *(Delay-Locked Loop)* Circuit utilisé pour stabiliser les signaux d'horloge dans les circuits intégrés.

DSP *(Digital Signal Processor)* Processeur spécialisé dans le traitement de signaux.

FPGA *(Field Programmable Gate Array)* Circuit intégré programmable, permettant un temps de développement plus court mais des performances et un niveau d'intégration moindres qu'un ASIC.

IP *(Intellectual Property)* Se dit de modules fonctionnels vendus par une société en possédant les droits d'utilisation.

PE *(Processing Element)* Élément de traitement d'un DSP. L'arrangement de plusieurs PE fonctionnant simultanément donne les architectures parallèles.

PULSE *(Parallel Ultra-Large Scale Engine)* Projet de DSP à architecture massivement parallèle, développé en vue du traitement d'images vidéo.

RAM *(Random-Access Memory)* Type de circuit de mémoire permettant les lectures et les écritures.

ROM *(Read-Only Memory)* Type de circuit de mémoire ne permettant que les lectures.

SIMD *(Single Instruction Multiple Data)* Une seule instruction pour plusieurs données. Type d'architecture parallèle employée dans les DSP.

VHDL *(Very high density Hardware Description Language)* Langage de programmation permettant la représentation de circuits logiques et la synthèse de ceux-ci.

INTRODUCTION

De nos jours, les microprocesseurs sont présents dans presque tous les aspects de la vie quotidienne. Bien sûr, l'application la plus flagrante est sans doute le microprocesseur que l'on retrouve au cœur de tout ordinateur, mais il existe un autre type de processeur encore plus courant : le processeur embarqué. Celui-ci est optimisé et spécialisé pour une tâche précise. Par exemple, une voiture moderne possède une puissance de calcul équivalente à six processeurs Pentium, pour contrôler une foule de paramètres allant du fonctionnement du climatiseur à l'admission d'essence au moteur, en passant par l'activation des freins antiblocages.

Les domaines d'application des processeurs spécialisés ne cessent de s'élargir avec le développement des nouvelles technologies de l'information. Pour réaliser la transmission et l'interprétation des données à des débits de plus en plus rapides, on a recours à des processeurs de traitement de signaux (DSP). En fait, la majorité des applications comportant un flot important de données à traiter font appel actuellement à de tels types de processeurs. La popularité des DSP se reflète sur l'évolution du marché actuel, illustré à la Figure I.1, qui affiche le taux de croissance le plus important à l'intérieur du marché des processeurs embarqués (Market Background. Lexra, Inc.).

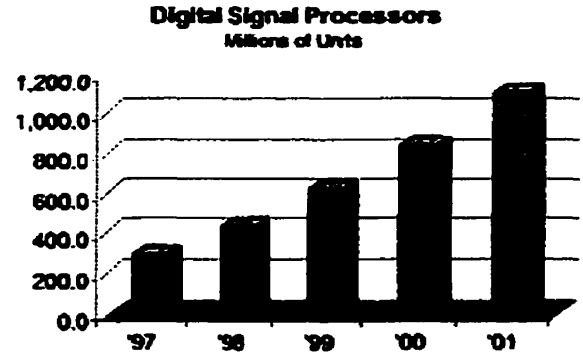

Figure I.1 Évolution du marché des DSP (*Market Background. Lexra, Inc.*)

Afin d'offrir des processeurs de mieux en mieux adaptés aux applications pour lesquelles ils sont destinés, certaines compagnies ont fait un pas en avant en introduisant des processeurs configurables. De plus, parmi ces entreprises, certaines vont même jusqu'à laisser toute la liberté de développement à leurs clients en fournissant des outils de développement qui leur permettent de configurer leur propre processeur sur mesure, à partir des options disponibles. La majorité de ces solutions se présentent sous forme de modules IP[1]. Ces processeurs configurables comportent plusieurs avantages notables sur leurs cousins à architecture fixe. On peut entre autres mentionner qu'ils sont sensiblement plus compacts et plus rapides, offrent plus de flexibilité de liaison avec d'autres modules, consomment moins d'énergie, permettent l'ajout de fonctionnalités et d'instructions spécifiques et ils sont compatibles avec une grande variété de technologies

[1] Un module IP ou « Intellectual Property Core » est un module fonctionnel conçu pour s'insérer dans une puce avec d'autres modules ou circuits logiques, dédiés à une application.

d'implantation lorsque distribués sous forme de modules IP. Finalement, tous ces avantages en font des candidats idéaux pour les applications de systèmes complets sur un seul circuit intégré (*system-on-chip*).

Avec la flexibilité grandissante qu'offrent aujourd'hui les manufacturiers de circuits intégrés, de nouvelles stratégies de développement de processeurs ont vu le jour. En effet, il est maintenant possible d'avoir recours à des circuits intégrés programmables comme des FPGA (*Field Programmable Gate Array*) pour la réalisation de ces derniers. Les FPGA actuels permettent un haut niveau d'intégration, en plus d'être programmables en quelques fractions de secondes. De plus, la majorité des modèles qui sont disponibles actuellement peuvent même être reprogrammmés à volonté. L'avantage qu'ont les FPGA au niveau de la rapidité de réalisation des circuits par rapport aux circuits dédiés ASIC (*Application Specific Integrated Circuit*) est énorme, compte tenu du délai de plusieurs semaines nécessaire à la réalisation de ces derniers. L'utilisation des FPGA permet donc un déverminage beaucoup plus efficace grâce aux corrections qui peuvent être rapidement apportées au circuit. L'aspect reprogrammable de ces circuits est aussi intéressant du point de vue de l'évolutivité, car les modifications apportées dans les versions subséquentes d'un produit pourront être incorporées dans les versions initiales. Cependant, l'avantage le plus marqué demeure un délai de mise en marché qui est grandement diminué comparativement aux ASIC, ce qui peut être déterminant pour le succès commercial d'un produit.

Ce mémoire expose les grandes lignes du développement et de l'optimisation d'un chemin de données configurable, faisant partie intégrante d'un processeur DSP. Le point de départ de ce travail est un modèle VHDL synthétisable du processeur PULSE (*Parallel Ultra-Large Scale Engine*), initialement développé pour être réalisé dans un circuit ASIC (Marriott et al., 1998). Ce processeur est de type SIMD (*Single Instruction Multiple Data*), ce qui signifie qu'il est composé principalement d'une matrice d'éléments de traitement ou PE (*Processing Elements*) effectuant tous les mêmes opérations simultanément sur des données différentes. C'est donc une architecture de calcul hautement parallèle, adaptée principalement pour effectuer du traitement numérique d'images.

Le modèle VHDL a dû être modifié et optimisé en vue d'une implantation dans un FPGA de la famille Virtex de Xilinx (Virtex product specifications. Xilinx, Inc.). Toutefois, la majeure partie de ce travail consiste à rendre le chemin de données configurable. Ainsi, une série de paramètres permet de contrôler différents aspects de l'architecture lors de la synthèse, tels que la largeur des mots traités, la profondeur des mémoires et des registres, la configuration de pipeline du multiplicateur et le nombre de PE du chemin de données. Cependant, la propriété la plus intéressante est la possibilité d'adapter le chemin de données à une application spécifique, en incluant ou en excluant divers modules composant les PE. Ainsi, le circuit généré lors de la synthèse du code VHDL sera optimisé exclusivement pour l'application visée, ce qui permet, tel que vu précédemment, un circuit plus compact. La complexité du PE étant diminuée, deux options se présentent

alors au concepteur : le choix d'un FPGA plus compact et moins dispendieux ou la génération d'un plus grand nombre de PE dans le chemin de données, ce qui apporte une plus grande capacité de traitement et donc, un processeur plus rapide.

D'autres modifications touchant l'architecture ont été apportées au modèle afin d'obtenir une meilleure exploitation des ressources disponibles à l'intérieur des PE. En ajoutant des bus et des connexions supplémentaires, la communication entre les différents modules devient plus flexible et permet ainsi de simplifier la tâche de programmation du processeur. Finalement, la dernière étape consiste à valider la fonctionnalité du nouveau modèle de chemin de données afin d'y apporter les correctifs nécessaires.

Le premier chapitre de ce mémoire est une revue de littérature ayant pour but d'exposer les différents travaux ayant été effectués à ce jour dans le domaine des processeurs configurables. Différentes méthodologies de conception ainsi que plusieurs exemples de réalisations y sont exposés. La revue se termine par une introduction aux processeurs et circuits reconfigurables dynamiquement, c'est-à-dire de façon simultanée avec l'exécution d'un programme par le processeur.

Le second chapitre présente un algorithme de génération de multiplicateurs signés, pipelinés et configurables. Selon les paramètres d'entrée, les multiplicateurs générés peuvent traiter des données de tailles diverses et comporter des configurations de pipeline variées, engendrant différentes latences. Cet algorithme, une fois programmé, fait partie

du modèle VHDL global du chemin de données configurable. Ce cas particulier est traité séparément des autres modules, étant donné la complexité de sa réalisation et son intérêt dans le contexte de différents circuits configurables et reconfigurables dynamiquement.

Le troisième chapitre présente l'architecture PULSE utilisée comme point de départ dans ce travail. Une attention particulière est accordée au chemin de données étant donné le fait qu'il constitue l'objet principal de ce mémoire. La technologie d'implantation ciblée, la famille de FPGA Virtex de Xilinx, est ensuite exposée en expliquant différentes caractéristiques intéressantes pour notre réalisation. Le chapitre se termine par la description des différents outils de développement utilisés.

Le quatrième et dernier chapitre constitue le cœur de cet ouvrage en proposant une méthodologie de développement pouvant être appliquée lors de la transformation de circuits conventionnels en circuits configurables. Toutes les étapes encourues pour la transformation du chemin de données sont décrites en détails, en partant du modèle original jusqu'au modèle final. La stratégie de validation mise en œuvre est explicitée ainsi que différents résultats visant à mettre en évidence l'impact de la modularité du chemin de données sur la complexité et la performance du circuit résultant.

CHAPITRE 1

REVUE DE LITTÉRATURE

Ce chapitre a pour but d'exposer brièvement au lecteur les différents travaux réalisés dans le domaine des processeurs et chemins de données configurables. La première partie de cette revue de littérature couvre les principaux produits offerts actuellement sur le marché des processeurs configurables. Quelques spécifications importantes ainsi que les outils de développement et les différentes options au niveau de la conception sont exposés. Les méthodes mises en œuvre pour la création et la validation de ces processeurs configurables ne sont cependant pas dévoilées publiquement par les sociétés en question. La seconde partie du chapitre traite de certaines méthodologies développées en vue de rendre modulaires certains designs existants, ainsi que des procédures proposées pour l'implantation de ces circuits. De plus, cette partie montre plusieurs prototypes réalisés dans différents domaines d'applications. La revue de littérature se termine par un survol des circuits à reconfiguration dynamique qui, bien que peu répandus pour l'instant, permettent déjà d'entrevoir des performances intéressantes, couplées à des avantages considérables sur les circuits conventionnels.

Il est opportun ici d'expliquer la différence entre certains termes employés dans cet ouvrage. Premièrement, il faut faire la nuance entre les circuits configurables et les circuits reconfigurables. Les premiers permettent à un utilisateur de créer un circuit optimisé et adapté à une application donnée sans toutefois pouvoir modifier ce circuit une

fois implanté. Les circuits reconfigurables ont cependant cette flexibilité accrue de pouvoir modifier le circuit à plusieurs reprise selon les utilisations qu'on compte en faire. De plus, il existe deux types de circuits reconfigurables. La section 1.6 aborde ce sujet en définissant la reconfigurabilité statique comme étant une seule configuration par application, contrairement à la reconfigurabilité dynamique qui implique des modifications au circuit pendant l'exécution d'une application. On peut ajouter que lorsqu'elle n'est pas spécifiée, c'est généralement de la reconfigurabilité statique dont on parle.

1.1 Processeur ARC

La compagnie ARC (ARC Cores, Ltd) fut la première à offrir commercialement un processeur sous la forme d'un module IP hautement configurable. Celui-ci comporte une architecture RISC (*Reduced Instruction Set Computing*) de 32 bits, entièrement nouvelle et développée exclusivement pour la flexibilité de configuration. Il peut atteindre des fréquences d'opération avoisinant 200 MHz lorsque implanté avec une technologie CMOS de 0,18 micron. Sa complexité est estimée à 16 000 portes logiques pour une configuration typique. Un des facteurs que la compagnie a su privilégier lors de la conception de ce processeur est la faible consommation d'énergie. En effet, parmi les applications visées pour un tel processeur, on retrouve plusieurs appareils mobiles ou portatifs qui nécessitent une alimentation à pile. Une autre propriété intéressante est la possibilité d'interconnecter plusieurs processeurs ARC ensemble, ce qui est intéressant

pour les applications intensives, afin d'obtenir une plus grande capacité de traitement. Par exemple, certains concepteurs ont déjà fabriqué des designs fonctionnels comportant 32 processeurs regroupés sur un seul circuit intégré.

La compagnie ARC affirme que son processeur peut être utilisé avec trois niveaux de configuration de plus en plus évolués. Le premier ne fait appel à aucune configuration de la part du concepteur qui peut utiliser directement le processeur ARC comme tout autre machine RISC de 32 bits. Le second consiste à sélectionner, dans une liste, différentes modifications et des modules variés qui se grefferont à l'architecture de base lors de la synthèse. Cette méthode offre un bon niveau de flexibilité tout en demeurant facile à utiliser, car toutes ces options et leur intégration dans le modèle général ont été préalablement optimisées et validées par la compagnie ARC. Le troisième et dernier niveau de configuration est rendu possible grâce à la disponibilité du code VHDL pour les concepteurs. En fait, ils ont la possibilité de modifier eux-mêmes le modèle VHDL synthétisable du processeur, afin d'y inclure les fonctionnalités nécessaires. ARC est d'ailleurs la seule société qui distribue présentement son processeur sous cette forme. Sa philosophie fait la promotion d'une démocratisation de la phase de conception, car elle juge qu'elle ne connaît pas à fond toutes les exigences des clients et qu'ils sont les mieux placés pour créer un processeur vraiment adapté à leurs besoins.

Les options disponibles pour le second niveau de configuration sont nombreuses et de nouvelles sont régulièrement ajoutées par la compagnie. Voici cependant les principales options disponibles :

- Synthèse en vue d'une faible complexité ou d'une fréquence élevée
- Sélection de la mémoire cache d'instructions
- Sélection des instructions requises à l'application
- Taille des registres
- Ajout de registres à usages spécifiques
- Choix de divers codes de condition
- Inclusion de mémoires RAM tampon localisées
- Bibliothèque d'extensions multimédia pour les applications 3D
- Fonctions pour le traitement de signal

L'outil de travail utilisé pour la configuration du processeur se nomme ARChitect et se présente sous la forme d'une interface graphique simple à utiliser. Le concepteur peut choisir parmi plusieurs bibliothèques d'options disponibles selon le type de processeur visé. L'outil donne un estimé du nombre de portes logiques totales et de la fréquence maximale d'opération pour chaque ajout ou retrait d'options sélectionnées. Les autres outils de développement disponibles comprennent de la documentation, une interface de simulation et un compilateur pour le langage C, permettant la simulation du code et son déverminage. À ces outils logiciels s'ajoute un système de prototype, basé sur des FPGA

reprogrammables, utilisé pour simuler de façon réelle le processeur avant la phase finale d'implantation.

1.2 Processeur Xtensa

La seconde compagnie majeure dans le domaine des processeurs configurables, Tensilica (Tensilica, Inc.), vend aussi son produit sous forme de module IP. Celui-ci est basé sur une architecture de 32 bits, optimisée pour la configuration et entièrement propre à Tensilica. Sa fréquence maximale d'opération est spécifiée à 250 MHz pour une technologie d'implantation CMOS de 0,25 micron, grâce à un pipeline à cinq niveaux. Un modèle type du processeur Xtensa comporte environ 25 000 portes logiques. Tout comme la compagnie ARC, Tensilica favorise une faible consommation énergétique pour son processeur. Cependant, elle a fait un pas de plus en implantant une fonctionnalité qui coupe dynamiquement l'alimentation des portions de circuit inutilisées.

Le processeur Xtensa est distribué en librairies de cellules disponibles pour différentes technologies. En effet, la compagnie Tensilica opte plutôt pour s'assurer que toutes les configurations possibles pour son processeur sont valides, en empêchant le troisième niveau de configuration alloué par ARC. Elle pense aussi qu'en voulant accélérer une partie du processeur, l'utilisateur peut en ralentir une autre car il ne connaît pas profondément l'architecture du processeur. Tensilica offre donc des outils qui ajustent

automatiquement les paramètres critiques selon les configurations choisies, au détriment d'un niveau de configuration supérieur.

Voici maintenant un aperçu des différentes options disponibles pour configurer le processeur Xtensa. Tensilica ajoute régulièrement de nouvelles possibilités et introduira des fonctions pour les calculs en virgule flottante.

- Résultat en fonction de la fréquence et de la complexité
- Inclusion d'un multiplicateur 16 bits
- Inclusion d'une unité de multiplication-sommation 16 bits
- Taille des registres de 32 ou 64 mots
- Différentes architectures de mémoire
- Support de signaux d'interruption
- Test par interface JTAG
- Fonctions pour le traitement de signal

Les deux outils principaux pour la création de processeurs dédiés comprennent un générateur de processeur et un environnement de développement logiciel. Le premier s'apparente à l'outil ARChitect et permet au concepteur de sélectionner les différentes options du processeur de façon graphique, tout en affichant aussi une approximation de la fréquence d'opération et du nombre de portes nécessaires pour la technologie visée. Il peut même analyser automatiquement le code C de l'application et optimiser le

processeur en conséquence. Des gains de performance variant de 5 à 50 fois ont été observés pour l'utilisation de cette fonction. L'environnement de développement logiciel génère tous les outils de programmation nécessaires adaptés au processeur créé. Ceci comprend un compilateur C/C++, un assembleur et différents outils de simulation et de déverminage. Un système de prototype est aussi inclus pour les simulations matérielles.

1.3 Comparaison et applications

Ces deux produits sont présentement les plus évolués sur le marché et offrent une suite intéressante d'outils de développement. Un aspect important pour la flexibilité est la portabilité qu'offrent ces processeurs, particulièrement pour celui de ARC distribué en VHDL. Un article (How Much Configurability Do You Need? Issues Forum, Silicon Stategies) soulève plusieurs points importants en présentant une conférence entre les représentants des compagniesARC et Tensilica. Les avantages et inconvénients des processeurs configurables sont exposés, ainsi que plusieurs autres aspects situant la position de ces produits parmi les processeurs en général.

Bien que plusieurs réalisations effectuées à l'aide des processeurs configurables touchent des applications domestiques telles que les caméras numériques, les imprimantes et les systèmes de télévision numériques, le domaine des communications est présentement celui qui semble le mieux adapté à l'utilisation de ces processeurs. Due à l'émergence continuelle de nouveaux protocoles, à la mise en marché qui doit se faire très rapidement

et aux mises à jour fréquentes, les concepteurs adoptent rapidement ces processeurs qui leur offrent un niveau de flexibilité inégalé. Contrairement aux processeurs traditionnels qui ne possèdent qu'un jeu d'instruction basé sur des opérations logiques et arithmétiques, les processeurs de ARC et Tensilica rendent possible la création d'instructions dédiées aux communications, telles que « extraction et décodage d'en-tête ». Cette nouvelle possibilité permet une meilleure utilisation des ressources matérielles mais surtout, elle permet à une compagnie d'implanter des algorithmes spécifiques confidentiels directement sous forme d'instructions, sans crainte de voir la concurrence récupérer ces algorithmes alors implantés de façon matérielle.

En intégrant ces processeurs dans des circuits programmables à haute densité, certains périphériques et coprocesseurs peuvent être inclus à même le circuit, afin de supprimer les bus de communication. Cette approche permet de meilleures performances tout en supprimant plusieurs problèmes reliés à l'utilisation de bus comme les délais de transfert, les protocoles de synchronisation ou l'allocation de bande passante.

1.4 Méthodes de conception de processeurs configurables

Bien que ARC et Tensilica gardent confidentielles les idées et les procédés mis en œuvre lors de la conception de leurs processeurs configurables, certains articles ont été publiés à ce sujet et proposent différentes méthodes à adopter pour la confection de tels processeurs.

1.4.1 Modules nécessaires et optionnels

La première méthode (Dyck et al., 1999) propose simplement l'idée que pour offrir un niveau de flexibilité suffisant, un processeur configurable doit permettre certaines variations au modèle de base et comporter une banque de modules optionnels pouvant être intégrés à l'architecture. Les options proposées sont résumées dans le Tableau 1.1.

Tableau 1.1 Modules configurables

Module	Type	Choix possibles
registre à usage général	nécessaire	4-64 mots
mémoire de programme	nécessaire	16-512 mots
registre d'état	nécessaire	4-32 bits
mémoire	optionnel	au besoin
mémoire FIFO (*First-In-First-Out*)	optionnel	au besoin
bus PCI	optionnel	taille des mémoires tampon
unité CRC (*Cyclic Redundancy Checker*)	optionnel	selon performance visée
unité MAC (*Multiply-Acumulate*)	optionnel	selon performance visée

L'article présente également une stratégie de communication entre processeurs où la transmission de données se fait par les registres à usage général. Deux processeurs échangent des données en intervertissant une portion de leurs registres en un cycle d'horloge.

1.4.2 Flexibilité versus complexité

Pour un processus de configuration efficace, un bon compromis entre la flexibilité et la complexité architecturales doit être atteint (Smith et al., 1989). Un processeur configurable doit être composé de blocs fondamentaux ayant une granularité appropriée à la variété d'applications supportées. En effet, une porte NON-ET est un bloc fondamental avec une grande flexibilité mais sans complexité. En contrepartie, un PE possède une grande complexité au détriment de la flexibilité de configuration. Cet équilibre doit être planifié dès le début de la conception d'un circuit configurable afin de minimiser le temps de développement (complexité architecturale des blocs élevée) tout en conservant un bon niveau de configurabilité (flexibilité du circuit élevée).

1.4.3 Architecture basée sur les transferts de données à haute vitesse

Afin d'être en mesure de traiter les débits élevés requis par le transfert d'images vidéo, de voix ou de données à travers les réseaux, une architecture configurable a été proposée pour supporter plusieurs protocoles de communication existants (Grayver et Daneshrad, 1998). Les modes d'opération possibles comprennent le filtrage numérique FIR/IIR réel et complexe, le filtrage basé sur l'erreur quadratique moyenne minimale, les transformées discrètes de Fourier et la synthèse de fréquences numériques directes. Tous ces modes sont supportés par un seul chemin de données composé d'unités de calcul interconnectées selon des patrons spécifiques à chaque application. Cette unité fondamentale de calcul est

en fait le plus petit dénominateur commun des algorithmes supportés et elle est illustrée à la Figure 1.1.

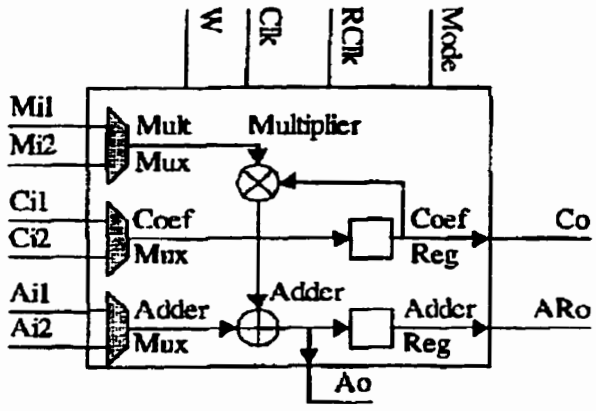

Figure 1.1 Unité de calcul fondamentale (*Grayver et Daneshrad, 1998*)

1.4.4 Utilisation de librairies hiérarchiques de modules VHDL

Cette approche implique la création de modules de base paramétrables et synthétisables en VHDL pour ensuite en faire l'organisation en banques hiérarchiques (McCanny et al., 1996). Chacun des modules d'une banque est composé de modules de la banque se situant au niveau inférieur. Cette méthode accélère considérablement le processus de conception car elle ne nécessite que l'assemblage de blocs entre eux selon la fonctionnalité désirée du processeur. La hiérarchie des banques est montrée à la Figure 1.2, où les banques du bas comprennent les modules de bas niveau et celles du haut, les modules fonctionnels de haut niveau.

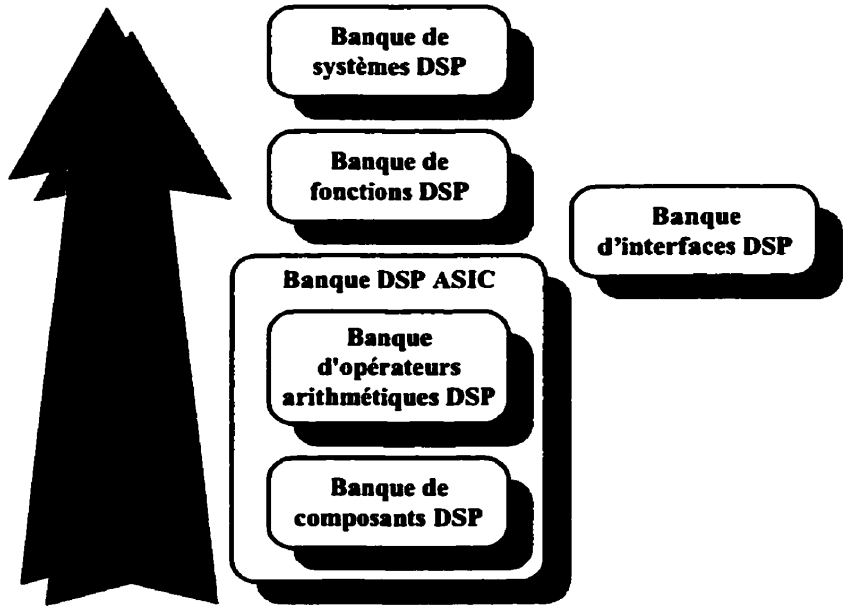

Figure 1.2 Hiérarchie des librairies hiérarchiques (*McCanny et al., 1996*)

Pour mieux illustrer le rôle de chaque banque, quelques exemples types des blocs que l'on retrouve dans celles-ci sont présentés dans le Tableau 1.2.

Tableau 1.2 Contenu des différentes banques

Banque de composants DSP	Banque d'opérateurs arithmétiques DSP
compteurs	multiplicateurs
comparateurs	accumulateurs
éléments de délai	multiplicateurs-sommateurs
convertisseurs de données	opérateurs de racines carrées
mémoires	diviseurs
décaleurs	sommateurs
Banque de fonctions DSP	**Banque de systèmes DSP**
filtres FIR	décodeurs MPEG
filtres IIR	décodeur JPEG
FFT	CODEC ADPCM[2]
filtres médians	H.261
décodeurs Reed-Solomon	PRML
décodeurs Viterbi	Détection et reconnaissance d'objets
Banque d'interfaces	
contrôleurs de mémoire	
interfaces de microprocesseurs	
contrôleurs d'interface de bus	
contrôleurs graphiques	

- La banque de composants DSP comporte plus de 70 blocs dont la paramétrisation doit permettre de supporter les différents formats de données possibles (non-signées, complément à deux, virgule flottante...).

---

[2] Adaptive Differential Pulse Code Modulation

- La banque d'opérateurs arithmétiques DSP est composée de plus de 100 blocs optimisés. Le type de paramétrisation requis pour ceux-ci concerne principalement les structures arithmétiques utilisées (*carry-save, carry-look-ahead, carry-ripple...*).

- La banque de fonctions DSP nécessite une paramétrisation propre à chaque fonction (niveau de troncation, profondeur du pipeline, paramètres des filtres...).

- La banque de systèmes DSP se situe au sommet de la hiérarchie et regroupe les fonctionnalités de haut niveau destinant le processeur à une application spécifique. La paramétrisation possible de cette banque doit simplement adapter et optimiser le système choisi à l'application.

- La banque d'interfaces doit être en mesure de supporter les communications entre le processeur DSP conçu et les périphériques externes grâce à une série d'interfaces disponibles.

1.4.5 Synthèse de processeurs DSP par analyse d'algorithmes

Cette technique synthétise automatiquement un processeur DSP adapté spécifiquement pour une application en réalisant son algorithme (Ramanathan et al., 1999). Cet algorithme doit être représenté sous forme d'un graphe de flot de données et de contrôle.

L'outil de synthèse utilise alors ce graphe pour générer les structures de calcul requises pour l'exécution de l'application. Le processus de synthèse se décompose en trois grandes étapes :

1. **Identification des structures de calcul minimales**. Elles correspondent à des sous-graphes du graphe de flot de données et de contrôle de l'algorithme à implanter. Ces structures minimales sont ensuite converties en différents circuits de base permettant de réaliser la fonctionnalité demandée, tout en obtenant les performances visées. Dans le cas contraire, les structures de calcul sont modifiées en y appliquant des transformations affines équivalentes, afin de générer de nouveaux circuits de base. De telles modifications sont itérées jusqu'à l'atteinte des objectifs.

2. **Synthèse de l'architecture correspondant à l'algorithme**. Les circuits de base sont interconnectés de telle sorte qu'ils forment un circuit de haut niveau exécutant l'algorithme demandé. Cette étape est itérative et peut exiger un retour à la première étape.

3. **Synthèse des structures de contrôle**. Les structures de contrôle nécessaires à l'algorithme sont synthétisées pour l'intégration dans l'architecture précédemment générée. Certaines séquences de contrôle peuvent nécessiter des modifications à l'architecture globale pour leur implantation.

1.5 Exemples de designs configurables

Quelques processeurs ou portions de processeurs ont été développés à ce jour à des fins expérimentales. Bien que fonctionnels et offrant des caractéristiques de configuration intéressantes, aucun de ces designs n'est devenu un produit commercial en mesure de se mesurer à ceux des deux compagnies principales, ARC et Tensilica. Cette section présente des processeurs et des chemins de données ayant atteint divers niveaux de configuration.

1.5.1 Module VS_DSP

Le module VS_DSP (Kuulusa et al., 1997; Nurmi et Takala, 1997), s'apparente particulièrement avec le travail dont il est question dans ce mémoire. C'est un DSP avec une architecture à trois étages de pipeline, globalement composée d'un chemin de données, d'un contrôleur, d'un générateur d'adresses, d'une mémoire de programme et de deux mémoires de données. Tous ces éléments sont connectés à l'aide de trois bus. Plusieurs paramètres peuvent être modifiés, mais les plus intéressants sont la longueur de mots de données paramétrable de 8 à 64 bits, la longueur du microcode choisie avec un minimum de 32 bits, un nombre d'accumulateur compris entre 2 et 4, 3 différents types d'adressage et plusieurs variantes de décaleur, ALU (*Arithmetic and Logic Unit*) et multiplicateur-sommateur. Des instructions peuvent aussi être ajoutées au jeu de base. Une fois la configuration choisie par le concepteur, les outils de développement génèrent

le processeur sous forme de cellules adaptées pour une des technologies supportées, ce qui n'offre pas la flexibilité d'un modèle VHDL entièrement portable.

1.5.2 MetaCore

Un second exemple présente MetaCore (Jin-Hyuk et al., 1998), un module DSP configurable avec un jeu d'instructions extensible. Son architecture se compose de trois parties distinctes fonctionnant en parallèle : une ALU, un générateur d'adresses pour la mémoire et un contrôleur. Celles-ci peuvent être configurées pour la longueur des mots de données et d'adresses et la taille de la mémoire, des registres et de l'accumulateur. Par ailleurs, les instructions inutiles à une application peuvent être supprimées et d'autres plus utiles peuvent être ajoutées. Le processeur final est synthétisé à partir de code VHDL pouvant être utilisé pour la simulation. Un premier prototype de 16 bits, le MDSP16, implanté sur une technologie CMOS de 0,6 micron, affiche une performance de pointe de 50 millions d'instruction par seconde (*MIPS*).

1.5.3 Générateur de processeurs RISC configurables

Cet outil (Berekovic et al., 1998) génère différents processeurs RISC à partir de paramètres sélectionnés par le concepteur. Le niveau de configuration disponible permet entre autres la sélection des longueurs de mots du chemin de données et d'instructions pour des valeurs comprises entre 8 et 64 bits pour le chemin de données et entre 8 et 32 bits pour les instructions. Un multiplicateur-sommateur de 64 bits et un accumulateur

peuvent être intégrés au besoin, alors que le nombre et la capacité des registres sont ajustables. Finalement, la profondeur du pipeline peut varier entre 1 et 5 étages. Une particularité intéressante de ce générateur est qu'il produit des modules codés en VHDL directement synthétisable, ce qui apporte une grande portabilité. Le nombre de portes logiques utilisées pour son implantation dépend des options choisies et peut varier entre un minimum de 900 portes pour un processeur 8 bits sans pipeline et un maximum de 10 000 portes dans le cas d'un processeur 32 bits à 5 étages de pipeline et 8 registres. Les prototypes réalisés ont permis de constater une fréquence d'opération maximale de 100 MHz pour un procédé CMOS de 0,5 micron.

1.5.4 PINE

Le module DSP PINE (Be'ery et al., 1993) est un processeur 16 bits à trois niveaux de pipeline, spécialement conçu pour les applications portables nécessitant une gestion d'économie d'énergie, afin d'obtenir une plus grande autonomie. Cette caractéristique autorise trois modes de fonctionnement, allant de l'inactivité complète à la pleine activité, en passant par un mode de fonctionnement au ralenti. Le processeur est principalement composé d'un contrôleur, d'un générateur d'adresses et d'une unité de calcul. Cette dernière comporte un multiplicateur, une ALU, ainsi que deux accumulateurs. Bien que moins flexible que les processeurs précédents, l'architecture PINE permet différents niveaux de configuration au niveau de ses RAM, ROM et de ses types d'entrées/sorties.

1.5.5 RaPID

L'architecture RaPID (*ReconfigurAble PIpelined Datapath*) (Cronquist et al., 1999; Ebeling et al., 1997) est un chemin de données se basant sur les réseaux systoliques. Il comporte plusieurs cellules reliées sériellement qui peuvent être des ALU, des multiplicateurs, des décaleurs, des registres, des mémoires, mais aussi des modules de calcul spécialisés à un traitement de données en particulier (Figure 1.3). Certaines de ces cellules, comme les registres et les mémoires, peuvent être configurées par le concepteur. Cependant, la paramétrisation se situe principalement au niveau des interconnexions et du décodage d'instruction. En fait, c'est l'algorithme à implanter qui détermine la configuration du décodeur, des interconnexions entre les cellules et les délais à insérer aux sorties de celles-ci. Comme l'indique son nom, RaPID est un chemin de données hautement pipeliné et c'est pourquoi les sorties de données doivent être synchronisées selon les différentes phase du pipeline pour permettre une réalisation adéquate de l'algorithme. Il faut toutefois mentionner que, de par sa nature sérielle, cette architecture est exclusivement optimisée pour des calculs hautement répétitifs et de faible complexité. Finalement, la fréquence d'opération atteinte par l'implantation de ce chemin de données sur une technologie CMOS de 0,5 micron est de 100 MHz, pour l'équivalent d'environ 1,5 BOPS (*Billion of Operations Per Second*).

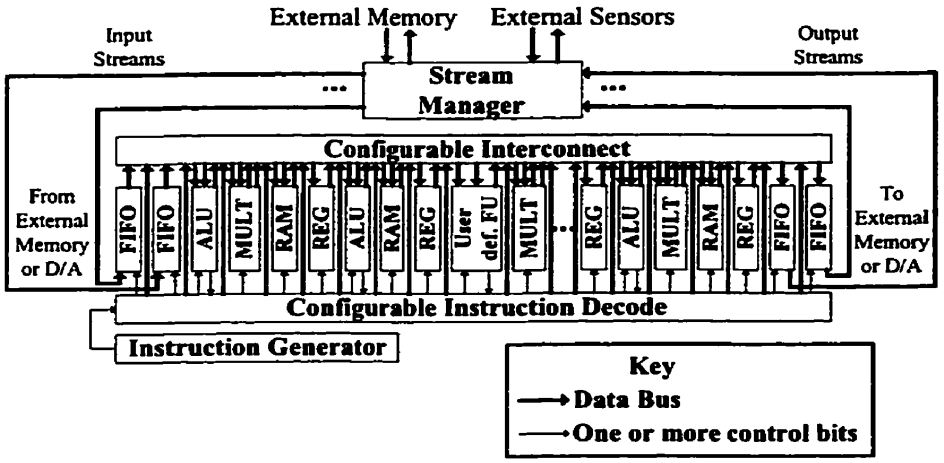

Figure 1.3 Architecture RaPID (*Cronquist et al., 1999*)

1.6 Reconfigurabilité

Nous terminerons cette revue de littérature en traitant des architectures reconfigurables, ce qui est justifiable par la popularité grandissante de cette méthode et les avantages qu'elle procure. La reconfigurabilité est définie comme étant la capacité de configurer une machine de façon répétée, pour effectuer différentes fonctions (Bhatia, 1997). Par opposition, le travail relaté dans ce mémoire, ainsi que tous les exemples de processeurs configurables cités précédemment, ne peuvent plus être modifiés une fois qu'ils ont été implantés. Cependant, la frontière entre ces deux types de circuits est mince, d'autant plus que l'utilisation de FPGA reprogrammables ouvre de nouveaux horizons aux concepteurs, en leur offrant une flexibilité supérieure (Fawcett, 1995). Il existe deux types principaux de reconfigurabilité : la statique et la dynamique.

1.6.1 Reconfigurabilité statique

La reconfigurabilité statique implique une seule configuration par application. En fait, une seule architecture spécialisée est employée pour la réalisation complète d'une tâche à effectuer. Ce n'est qu'une fois l'application terminée que le circuit peut être modifié de nouveau, afin d'être mieux adapté et optimisé à une différente tâche à effectuer. La reconfigurabilité statique est utilisée présentement par certains constructeurs électroniques pour leur permettre de modifier leurs circuits afin d'évoluer vers de nouveaux algorithmes ou protocoles et ainsi garder leurs produits compatibles avec les nouvelles normes. D'ailleurs, le domaine spatial considère de plus en plus cette méthode pour le développement de sondes et de satellites reconfigurables à partir de la Terre (Matsumoto, 1999). En ayant cette particularité, les engins spatiaux pourront être réutilisables à plusieurs missions et réparables à distance dans le cas d'une erreur d'implantation de circuit. De plus, ils pourront même n'être configurés et programmés qu'une fois rendus en orbite, ce qui offre plus de souplesse dans les délais de conception.

L'idée de la reconfiguration à distance peut cependant être ramenée à l'échelle terrestre, à travers différents réseaux de communication existants. Bien qu'intéressante, cette manière de procéder peut entraîner de sérieux problèmes auxquels il faudra faire face éventuellement (Westfeld, 1999). En effet, l'utilisation de réseaux publics implique une visibilité accrue et donc une ouverture au piratage des données. Une société pourrait voir l'architecture de son circuit copiée par une société concurrente lors du transfert de celui-ci vers un FPGA distant. Ces données pourraient aussi être altérées afin d'en modifier la

fonctionnalité une fois l'implantation réalisée, un peu à la manière des virus informatiques actuels. Finalement, un autre problème risque d'apparaître avec la capacité grandissante des FPGA; les fichiers utilisés pour l'implantation de circuits complexes pourraient nécessiter des temps de transfert trop longs. Tous ces nouveaux obstacles devront être franchis en ayant recours à de nouvelles techniques d'encryption et de compression de données, apportant une sécurité, une fiabilité et une efficacité accrues.

1.6.2 Reconfigurabilité dynamique

La reconfigurabilité dynamique implique que des modifications à la configuration d'un circuit puissent être faites simultanément avec l'exécution d'une application. Ces changements peuvent affecter toute l'architecture ou seulement une portion. L'utilisation de la reconfiguration dynamique est encore relativement récente et peu répandue, étant donné sa difficulté d'implantation. Généralement, de tels processeurs sont utilisés sous forme de coprocesseurs, en conjonction avec une plate-forme stable qui n'est pas reconfigurable. Malgré sa complexité de réalisation, les avantages de la reconfigurabilité dynamique sont nombreux. Tel qu'illustré à la Figure 1.4, la flexibilité s'approche de celle des microprocesseurs programmables et la performance atteint celle des circuits dédiés qui ne sont optimisés que pour une seule application.

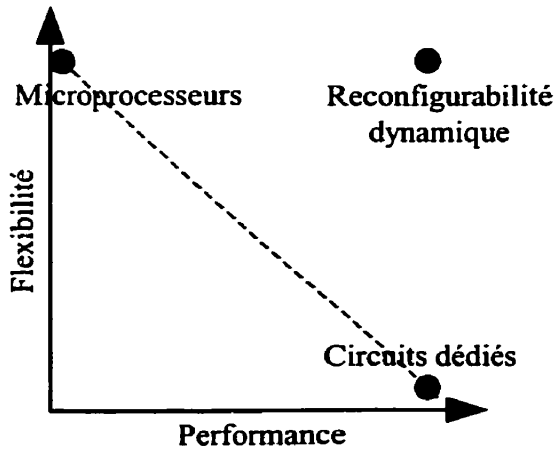

Figure 1.4 Avantages des architectures reconfigurables dynamiquement (Bhatia, 1997)

Une utilisation intéressante de ce type de reconfigurabilité fait appel à un seul FPGA pour émuler un design complexe destiné à être implanté sur un ASIC (Gajjalapurna et Bhatia, 1998). Un algorithme partitionne et réordonne temporellement l'architecture initiale pour ensuite adapter le contrôle des opérations. La nouvelle architecture fractionnée peut ensuite être implantée sur un FPGA de moindre capacité, tout en conservant la même fonctionnalité, en effectuant de la reconfiguration dynamique. Cette stratégie a permis la simulation d'un design de 75 000 portes logiques sur un seul FPGA de 13 000 portes. Il existe aussi différents exemples de processeurs et coprocesseurs implantés dans des FPGA reconfigurables dynamiquement : le nano-processeur (Wirthlin et al., 1994), Enable++ (Hogl et al., 1995), qui permet un gain en performance variant entre 100 et 1000 dépendant de l'algorithme implanté, et un coprocesseur spécialisé pour le traitement des bases de données du génome humain (Lemoine et Merceron, 1995), qui affiche aussi

une amélioration de deux à trois ordres de grandeur comparativement à un processeur conventionnel.

CHAPITRE 2

MULTIPLICATEUR CONFIGURABLE, SIGNÉ ET PIPELINÉ

Pour rendre un processeur configurable, plusieurs modifications sont nécessaires afin de rendre chacun des modules flexibles au niveau des paramètres sélectionnés. Certaines structures sont facilement adaptables alors que d'autres sont beaucoup plus complexes à rendre configurables. Dans le cadre de ce travail, le multiplicateur utilisé dans le chemin de données, fut le module qui nécessita le plus de modifications. La principale difficulté vient du fait que l'architecture du multiplicateur signé est étroitement liée à la longueur des mots de données traités.

Ce chapitre présente un article intitulé « A configurable generic two's complement pipelined multiplier » soumis à la revue « IEEE Design and Test of Computers » au mois de février 2000. Cet article présente la démarche employée pour développer un algorithme codé en VHDL, permettant la synthèse d'un multiplicateur aux caractéristiques voulues. Les paramètres d'entrée permettent la sélection de la longueur des mots de données et le nombre d'étages du pipeline ainsi que leur disposition. L'article décrit en détail l'architecture retenue et les relations mathématiques reliant celle-ci aux paramètres de configuration. Finalement, une étude comparative permet de situer le multiplicateur développé par rapport aux modèles existants dans les bibliothèques de modules arithmétiques de Xilinx.

A CONFIGURABLE GENERIC TWO'S COMPLEMENT PIPELINED MULTIPLIER

Alexandre Fortin, Yvon Savaria and Mohamad Sawan,

Department of ECE, Ecole Polytechnique de Montreal

Abstract -- In the context of reconfigurable processors, there is a need for generic multipliers that can compute operands of different bit-lengths and have a latency that can be adjusted to the environment. We propose in this paper an algorithm that can synthesize signed, pipelined and parallel multipliers optimized for any length greater than five bits. The Baugh-Wooley algorithm is used to process the sign. Pipeline depth and registers locations are also fully parameterizable. The algorithm has been successfully implemented in VHDL, tested, placed and routed for a state-of-the-art FPGA. Speed and complexity comparisons with fixed multipliers from Xilinx's library are made and demonstrate an increase in speed at the expense of a larger area.

2.1 Introduction

Nowadays, time-to-market is a very significant factor in the commercial success of a new product. A new device, even if it is greatly improved compared to existing products, could be a commercial failure if it does not reach the market on time. This principle is particularly relevant to electronic systems, where evolution is lightning fast. For this reason, hardware designers must complete their circuits quickly, without wasting

precious time in all design and test phases. To minimize the development time, designers have used different design methodologies and tools. One of such strategies is to have a generic architectural model, that the designers can adapt and modify to suit their needs (Levy, 1999). By using a customisable processor, developers are freed from the tasks of defining, implementing and validating a suitable architecture. This idea is growing in popularity in industry and some companies even restrict their activity to this field and provide hardware cores which can be adapted on demand.

Recent availability of high-performance field programmable gate arrays (*FPGA*), with densities in excess of one million system gates, providing dedicated datapaths for high-speed arithmetic and integrated high-bandwidth RAM (Virtex product specifications. Xilinx, Inc.), is a very interesting medium to implement embedded processors with flexible architectures. FPGAs have at least two major advantages over their application-specific integrated circuits (*ASIC*) counterparts. Firstly, they are programmable in a fraction of a second, which contributes to reduce turnaround compared to ASIC devices. Secondly, SRAM FPGAs are reprogrammable, and some systems even allow a dynamic reprogramming, that let some parts of the device be reconfigured while others are running. This flexibility allows a high level of optimisation never reached before, by having a chip whose architecture can be modified to best fit the requirements of a specific application.

At the current level of complexity, hardware description languages (*HDL*) are indispensable to design configurable processor cores. Not only do they significantly decrease the development time of new devices over a schematic design methodology, but they also make them easily configurable. It is even possible to modify an existing design to adapt it for another use. For example, a core is easily customizable for its data length, memories sizes, instructions set, pipeline depth, number of processing elements and for inclusion of custom modules. Languages like VHDL even allow a designer to use generic parameters, wherever needed in the code, to subsequently change the structure of the resulting circuit, just by changing their values and re-synthesising the code. Designs that have a regular architecture are easily made generic, while more complex designs, with irregular structures, cannot be adapted without a thorough rearrangement of their architecture.

Multipliers are common components found in almost every processor dedicated to execute mathematical functions or to perform signal processing. The architecture of these multipliers depends on their use, but also on the design goals. In fact, multipliers can be signed or unsigned and they can work with fixed-point or floating-point numbers. Furthermore, many hardware algorithms have been developed, each one offering a different compromise between speed and complexity. However, multiplier architectures are generally dependent on word length, especially with signed numbers. For that reason, generic variable length multipliers with a reasonable level of optimisation can be difficult to realise.

The algorithm proposed in this paper generates circuits to multiply two signed integers having the same number of bits represented in two's complement format. It can support any multiplicand and multiplier lengths greater than or equal to 6 bits. Also, the algorithm has the capability to insert registers at different places selected by the designer. As a result, the multiplier has a parameterizable latency. All these properties are very useful to designers, since they permit the use of this multiplier in different processor architectures, each one generally supporting a unique fixed latency and data width in their modules. Therefore, no deep modification in the rest of the architecture is required, which is then compatible with our main goal of providing a designer with a fast development time without extensive revalidation. The algorithm was implemented in VHDL 93, targeting the Virtex series of FPGAs from Xilinx (Virtex product specifications. Xilinx, Inc.).

2.2 Multiplier architecture

The architecture of the proposed multiplier is based on the Baugh-Wooley algorithm (Baugh and Wooley, 1973). The main advantage of using this algorithm is to generate only positive partial products, allowing to calculate the product by simply adding all the partial products. Also, it processes the signs simultaneously with the multiplication, without additional delay. As explained in the following, the multiplier structure is composed of a first stage of parallel two-bit multipliers, followed by a binary tree of adders to quickly execute as many additions as possible in parallel. In addition to the first

multiplication stage and the following tree of adders, the Baugh-Wooley algorithm requires an additional structure to calculate the sign of the result by producing a unique partial product, which is then added to the other partial products. This architecture is relatively fast, while maintaining a good regularity suitable for FPGAs and compatible with our objective of producing a generic length multiplier. Interesting work has been done on recursive generation of multipliers in VHDL (McCluskey, Practical Applications of Recursive VHDL Components in FPGA Synthesis) and, while signed multipliers are not as regular and cannot exploit easily their method, unsigned multiplier may be implemented that way to reduce the algorithm complexity.

For a signed multiplication, let $A = (a_{n-1}, \ldots, a_0)$ be the multiplicand, $B = (b_{n-1}, \ldots, b_0)$ the multiplier and $P = (p_{2n-1}, \ldots, p_0)$ the product. All these vectors are expressed using two's complement representation. When the multiplicand and the multiplier have the same length n, the Baugh-Wooley algorithm can be slightly simplified to generate one less partial product than the original algorithm. Figure 2.1 shows the multiplication procedure with all the partial products generated according to this reduced Baugh-Wooley algorithm. It should be noted that the first $n-1$ partial products correspond to the ones that would have been generated by the unsigned multiplication of A and B, without their most significant bits $(a_{n-2},\ldots,a_0) \times (b_{n-2},\ldots,b_0)$. The following three partial products are generated by the most significant bits a_{n-1} and b_{n-1} to take care of the sign in the two's complement representation. The first of these three partial products, $S1$, is formed by

producing the bit-wise inverse of (b_{n-2}, ..., b_0) and then multiplying the resulting vector by a_{n-1}:

$$S1 = (a_{n-1}b_{n-2}^*, a_{n-1}b_{n-3}^*, ..., a_{n-1}b_0^*) \quad (2.1)$$

The second partial product used for the sign, *S2*, is calculated by inverting (a_{n-2}, ..., a_0) and multiplying it by b_{n-1}:

$$S2 = (b_{n-1}a_{n-2}^*, b_{n-1}a_{n-3}^*, ..., b_{n-1}a_0^*) \quad (2.2)$$

The third partial product, *S3*, is a little more complex to generate and it is composed as follows:

$$s3_0 = a_{n-1} \; XOR \; b_{n-1} \quad (2.3)$$

$$s3_1 = a_{n-1} \; AND \; b_{n-1} \quad (2.4)$$

$$s3_{n-2}, ..., s3_2 = 0 \quad (2.5)$$

$$s3_n = s3_{n-1} = a_{n-1} \; OR \; b_{n-1} \quad (2.6)$$

Once the three partial products described above are computed, they must be added, and the resulting sum of *S1*, *S2* and *S3* is finally added to all the other partial products.

					b_{n-1}	b_{n-2}	... b_1	b_0	
					a_{n-1}	a_{n-2}	... a_1	a_0	
						a_0b_{n-2}	... a_0b_1	a_0b_0	
					a_1b_{n-2}	...	a_1b_1 a_1b_0		
				...	...	...	...		
			$a_{n-2}b_{n-2}$	...	$a_{n-2}b_1$	$a_{n-2}b_0$			
		$a_{n-1}b_{n-2}$*	...	$a_{n-1}b_1$*	$a_{n-1}b_0$*			$\Leftarrow$	*S1*
		$b_{n-1}a_{n-2}$*	...	$b_{n-1}a_1$*	$b_{n-1}a_0$*			$\Leftarrow$	*S2*
$a_{n-1}+b_{n-1}$	$_{n-1}+b_{n-1}$	0...	...0	$a_{n-1}\bullet b_{n-1}$	$a_{n-1}\oplus b_{n-1}$			$\Leftarrow$	*S3*
p_{2n-1}	p_{2n-2}	...	...	p_n	p_{n-1}	...	... p_1	p_0	

Figure 2.1 Partial products of a complete signed multiplication, with S1, S2 and S3 being used for signed representation

Processing independently the most significant bits of the multiplicand and the multiplier, and the reduced unsigned multiplication of the other $n-1$ bits, by independently generating the partial products before adding them, is a standard way to complete a multiplication. In the rest of this paper, all the real numbers are floored ($\lfloor x \rfloor$). This means that all the fractional values are ignored, as in VHDL where each number x is automatically truncated to the nearest integer smaller or equal to x.

The first step, shown in Figure 2.2, is realised using several two-bit multiplication modules having the same multiplicand (a_{n-2}, a_{n-3}, ..., a_0).

$$\text{Number of modules in first stage} = \lfloor n/2 \rfloor \tag{2.7}$$

Each module is a partial multiplier corresponding to $b_{2(i-1)+1}$, $b_{2(i-1)}$, i being the rank of the module, where $i=1$ corresponds to the multiplication module of the two least significant bits of B. The last module ($i = \lfloor n/2 \rfloor$) must have a different behaviour if n is even. In that

case, the last reduced multiplication processes an odd number of bits, and the last multiplication module has a multiplier equal to $b_{2(i-1)}$, with the most significant bit filled with zero. Note that all these modules can compute their results in parallel. The choice of using two-bit multiplications is dictated by a compromise between the complexity of computing the partial products and the tree of adders that follows. High speed multipliers are often pipelined, in which case the designer must distribute as equally as possible the total delay on all of the pipeline stages to obtain a balanced critical path. In our case, multiplication using two bit partial products was found to offer sufficient flexibility for inserting pipelining registers, while keeping a simple architecture.

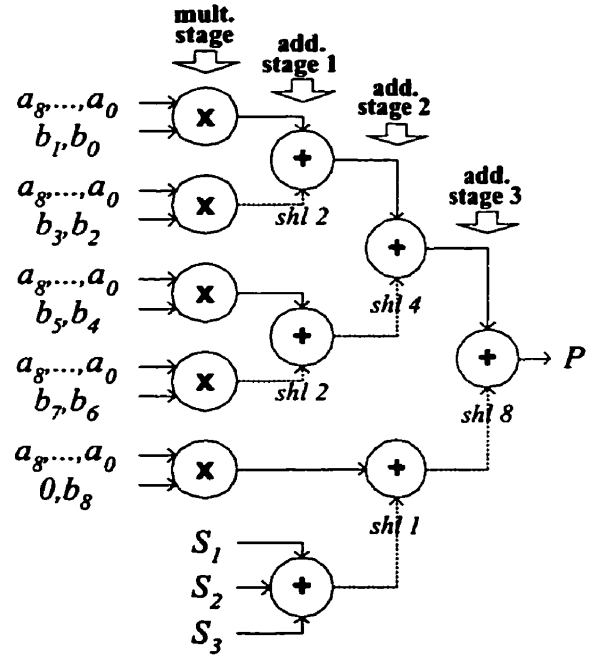

Figure 2.2 Schematic representation of a 10x10 multiplier, with *shl x* being a left-shift by x bits, and partial products *S1*, *S2* and *S3* being defined in equation (2.1) to equation (2.6)

The second step, after the generation of the partial products, is to add them in the most parallel way to form the product P, as illustrated in Figure 2.2. The required tree of adders is composed of $\lfloor log_2(n-2) \rfloor$ binary addition stages. Each addition stage is composed of $\lfloor (n + 2^{j+1} - 2) / 2^{j+1} \rfloor$ adders processing data in parallel, where j is the stage position number and $j=1$ is the first addition stage next to the multipliers. Hence, all adders use data coming from two arithmetic modules of the preceding stage. However, the last adder of a stage can occasionally have only one input when the number of arithmetic modules of the preceding stage is odd (bottom of Figure 2.2). In that special case, the second term of the adder does not exist in the VHDL description and there is no addition to perform. The input data is only propagated to the next stage.

All the partial products in the addition tree must be shifted to the left by a suitable number of bits, as shown in Figure 2.1. To realize this operation in a fast and efficient way, each addition stage must shift its second term 2^j bits to the left before summing it with the first one, where j is the stage position number. This shifting process is illustrated in Figure 2.2. Due to shifting, the bit-width of input and output signals vary according to the position. Hence, each adder stage receives input terms having a specific number of bits and produces output sums having a different number of bits. Both values are expressed in equations (2.8) and (2.9).

$$\text{Bit length of stage } j \text{ inputs} = n + 2^j - 1 \tag{2.8}$$

$$\text{Bit length of stage } j \text{ outputs} = n + 2^{j+1} - 1 \tag{2.9}$$

The bottom branch in the tree of adders of Figure 2.2 includes a three-input adder which is used to inject the result of the partial sum $S1 + S2 + S3$. In fact, this adder can also have two inputs instead of three when the number of modules in the preceding stage is odd. In this case, there is only one term to add to the sign term. The place selected to inject this partial sum allows to take advantage of the shifting done automatically by the last adder. Therefore, the remaining shift to do with the sign term is decreased to $n - 2^{j+1} - 1$ positions to the left.

However, computing the sign term in the tree of adders brings an additional challenge. In fact, it has been observed that for certain multiplier lengths n, adding the sign term increases the bit length of the sum. This occurs when each of the adders constituting the tree gets two input terms, producing a complete binary tree. To address this complication, we should consider that the result of the stage right before the last one can have two different lengths depending on the multiplier length n. Its output bit length is usually given by equation (2.9), as stated previously. However, for those special cases when the sign term affects the result length, the bit length of its output is given instead by equation (2.10).

$$\text{Bit length of the output} = 2n - 2^{j+1} \tag{2.10}$$

By always using the largest value among equations (2.9) and (2.10), the problem is solved for every possible values of n.

2.3 Algorithm

All the previously described characteristics have been integrated in an algorithm to synthesize generic multipliers. Executing it for a given length n allows to create a multiplier for n-bit signed integers. Once generated, a multiplier can only work with inputs having the specified length. To change the length or the pipelining, the proposed algorithm must be executed again with the new word length and pipeline parameters in order to generate a new circuit. Since the detailed design is completed by a fast synthesis tool, multipliers built by this algorithm can be generated on-demand to provide a reconfigurable core. The algorithm that produces a family of generic multipliers is described in the following pseudo-code implementation.

generation of the first stage which is composed of parallel multipliers

for all multipliers of the multiplication stage

    if not last multiplier of the stage OR n is odd

        generate a multiplier

    if last multiplier of the stage AND n is even

        generate a multiplier with null second term's MSB

generation of the subsequent stages which are composed of parallel adders

```
for all subsequent addition stages
    generation of the first addition stage
    if first addition stage
        for all adders in the stage
            if not last adder of the stage
                generate an adder
            if last adder of the stage AND not stage right before the last one AND number
            of multipliers in first stage is even
                generate an adder
            if last adder of the stage AND not stage right before the last one AND number
            of multipliers in first stage is odd
                propagate the term to the next stage
            if last adder of the stage AND stage right before the last one AND number of
            multipliers in first stage is even
                generate an adder with sign input
            if last adder of the stage AND stage right before the last one AND number of
            multipliers in first stage is odd
                generate an adder with one term and sign input
    generation of all the addition stages following the first addition stage
    if stage follows the first addition stage AND precedes the last one
        for all adders in the stage
            if not last adder of the stage
```

generate an adder

if last adder of the stage AND not stage right before the last one AND number of adders in preceding stage is even

generate an adder

if last adder of the stage AND not stage right before the last one AND number of adders in preceding stage is odd

propagate the term to the next stage

if last adder of the stage AND stage right before the last one AND number of adders in preceding stage is even

generate an adder with sign input

if last adder of the stage AND stage right before the last one AND number of adders in preceding stage is odd

generate an adder with one term and sign input

generation of the last addition stage of the adder tree

if last addition stage

generate an adder

2.4 Pipelining strategy

Pipeline construction is realized outside the algorithm. It is obtained by placing registers between consecutive stages chosen by the designer. In addition, registers can also be present at the main output of the multiplier to produce a valid result at the rising edge of

the clock. Therefore, positions of all registers are controlled by generic parameters that offer the flexibility to choose the pipeline configuration which is best suited for all different multiplier lengths n. Figure 2.3 illustrates all possible useful locations where pipeline registers can be placed for a 17x17 multiplier. By using these parameters, registers can be inserted or not between every stage to provide possible latencies from none, when no register is used, to $\lfloor log_2(n-2) \rfloor + 1$, when registers are placed between each stages and at the main output. To remain synchronized, the Baugh-Wooley section automatically adjusts its latency to the rest of the circuit. Again, registers should be located in a way to equalize delays between pipeline stages and the optimal solution is application dependent.

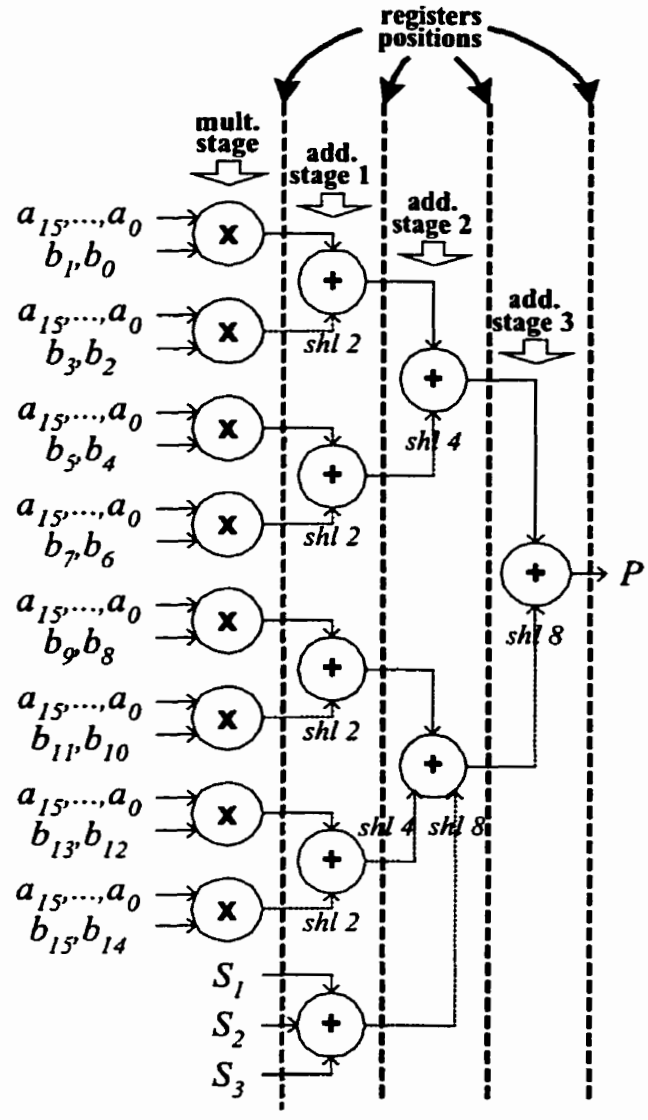

Figure 2.3 Schematic representation of a 17x17 pipelined multiplier. Dotted vertical lines indicate all the useful positions where pipeline registers can be inserted

Another possibility is to insert registers inside adders instead of between them. The advantage of using this method is to break carry chains inside adders of a stage to reduce

the carry propagation delay. This delay is the intrinsic critical path of each conventional adder such as the ones used in this design. However, the target FPGA technology possesses built-in fast carry chains dedicated to arithmetic that limit the carry propagation delay of each adder. The need to insert pipeline registers inside adders is then reduced and we chose not to implement them since it would substantially complicate the algorithm and the VHDL code. To confirm the validity of this choice, experiments were made for simple adders of different lengths. It consists of measuring the highest frequency attainable by two adders of the same length; one completely combinational and the other having a pipeline register in the middle of its carry chain. The results are compiled in Table 2.1.

Table 2.1 Performances (MHz), in a Xilinx XCV800 –5 FPGA, for combinational and pipelined carry chain of different lengths adders

Adder Length	Maximum frequency (MHz)	
	Combinational	Pipelined
8x8	198.26	204.08
10x10	195.12	200.60
12x12	190.40	195.39
16x16	186.19	187.30
20x20	169.43	171.85
24x24	162.92	159.26
32x32	154.78	152.74
48x48	138.27	149.66
64x64	123.17	132.21
96x96	98.63	106.39
128x128	53.86	83.86

These results show that performances in both configurations, for adders lengths of 32 bits or less, are very similar. In fact, below this length, the fast carry chains present in Virtex FPGAs eliminate the need for inserting registers inside adders. In Table 2.1, the average percentage difference in frequency between combinational and pipelined adders is 7.82 % but it reduces to 0.98 % if we consider lengths of 32 bits and less. Pipelining adder modules becomes of interest only above 96 bits. Since the lengths of multipliers generated by the proposed algorithm is likely to be less than that, it is justified to keep the present pipelining strategy by inserting registers between adders instead of inside them, and thus keeping the algorithm simpler.

2.5 Results

The proposed algorithm has been implemented in VHDL and was successfully tested. A difficulty comes from the fact that "generate" statements in VHDL code do not handle the "else" clauses with if declarations (Ashenden, 1996). For that reason, the algorithm does not use any "else", but one can easily modify it by adding "else" declarations at the right places if another target language supports "else" statements.

Several software tools were used to complete an efficient implementation of the algorithm. The first one, Active VHDL 3.3, helped the development and the validation of the code. Synplify 5.1.5a was used to synthesise this code while supplying an architectural view of the final design generated during the synthesis. Finally, the multipliers library developed by Xilinx for its Virtex FPGAs family was used for comparison purposes. In addition, the mapping, the placing and routing as well as the timing analysis tools supplied in Xilinx Alliance 2.1a (Alliance Series v2.1I Software. Xilinx, Inc.) were used. The following results were produced with Alliance which provides accurate timing analysis reports. The software versions used in this work were the most recent available at the time of completing this paper.

The two major parameters analysed to evaluate and compare different multiplier designs are the speed and the area complexity. More specifically, the speed is the maximum operational frequency allowed by the design for the specified FPGA device. A speed

grade 5 XCV800 Virtex FPGA was assumed. The complexity is estimated by the number of slices used to implement the design. Each FPGA is composed of an array of configurable logic blocks (*CLBs*) where each block is constituted by two slices. Another meaningful measure of the complexity is the fraction of the device used to implement the multiplier. It allows to estimate how many multipliers of that size could be used in the selected FPGA.

In order to select an adequate placement of the pipeline registers, we first studied its influence on performance versus multiplier size. The results obtained for all possible registers configurations with three different multiplier lengths are shown in Table 2.2. The convention used to represent different pipeline configurations is a binary number, where each digit corresponds to a register position, with 1 indicating the presence of registers and 0 its absence. The most significant bit represents the registers position between the multipliers stage and the first adders stage. All the other digits follow intermediate adders stages in the same way, to finally have the least significant bit corresponding to the position between the stage right before the last one and the last stage. These results demonstrate that for multipliers having the same length and the same latency, the performance may vary substantially depending on the pipeline configuration adopted. Somewhat surprisingly, the optimal position is not easily predictable, probably due to variability resulting from the placement and routing process. A feasible approach is to synthesize all solutions for a desired multiplier latency and select the one producing the best results.

Table 2.2 Performances (MHz) for all pipeline configurations of three different lengths multipliers. Binary numbers represents configurations, where each bit indicates the presence (1) or absence (0) of registers at corresponding position. Results are grouped by identical latencies

Maximum frequencies (MHz) for every pipeline configurations of a 8x8 multiplier. Results grouped by latency.							
00	58.469	***01***	78.015	***10***	85.077	***11***	91.937

Maximum frequencies (MHz) for every pipeline configurations of a 16x16 multiplier. Results grouped by latency.							
000	50.241	***001***	61.550	***010***	82.041	***100***	64.041
011	80.645	***101***	84.955	***110***	87.727	***111***	110.16

Maximum frequencies (MHz) for every pipeline configurations of a 20x20 multiplier. Results grouped by latency.							
0000	40.945	***0001***	57.379	***0010***	69.339	***0100***	65.062
0011	70.437	***0101***	80.658	***0110***	77.942	***1000***	63.674
0111	80.160	***1001***	74.184	***1010***	81.793	***1100***	70.166
1011	87.116	***1101***	78.168	***1110***	85.616	***1111***	95.039

To measure the performance of the synthesized multipliers with the proposed procedure, we compared them to other similar signed multipliers available in a library produced by Xilinx (Core Solutions – Base-Level Products. Xilinx, Inc.). These designs were not

produced by the Xilinx Core Generator which was not yet available for Virtex devices. Results are summarized in Table 2.3. For each specified length, a multiplier was generated using the proposed generic algorithm, and we have also reported the performances obtained from the supplied specifications for one or more comparable reference pipelined multiplier designs.

Table 2.3 Comparison between generic length multipliers and multipliers from Xilinx's "Reference Designs" library synthesized for a Xilinx Virtex XCV800 -5

Length	Multiplier	Latency	Frequency (MHz)	Complexity	
				(Slices)	(%)
6x6	generic	0	59.056	42	0.45
	generic	2	95.048	48	0.51
8x8	generic	0	58.469	66	0.70
	generic	1	85.077	77	0.82
	m0808sr1	1	77	39	0.41
	generic	2	91.937	73	0.78
	m0808sr4	4	160	48	0.51
10x10	generic	0	56.847	94	1.00
	generic	3	113.934	123	1.31
	m1010sr5	5	142	79	0.84
12x12	generic	0	55.084	125	1.33
	m1212sc	0	51	82	0.87
	generic	1	82.041	133	1.41
	m1212sr1	1	57	86	0.91
	generic	3	110.084	150	1.59
	m1212sr5	5	143	107	1.14

14x14	generic	0	51.464	158	1.68
	generic	3	115.009	198	2.10
16x16	generic	0	50.241	203	2.16
	m1616sc	0	46	143	1.52
	generic	3	110.156	234	2.49
	m1616sr5	5	139	168	1.79
18x18	generic	0	48.544	248	2.64
	generic	4	109.926	336	3.57
	m1818sr6	6	107	222	2.36
20x20	generic	0	45.185	296	3.15
	generic	4	95.039	391	4.16
22x22	generic	0	48.144	350	3.72
	generic	4	79.519	449	4.77
24x24	generic	0	43.527	405	4.30
	generic	4	90.188	496	5.27
26x26	generic	0	39.620	460	4.89
	generic	4	73.057	587	6.24
28x28	generic	0	37.410	524	5.57
	generic	4	75.239	648	6.89
30x20	generic	0	40.104	592	6.29
	generic	4	75.154	725	7.71
32x32	generic	0	40.619	677	7.20
	generic	4	69.295	796	8.46

Interpreting these results is not an easy task due to differences in latency of some compared multipliers. In fact, the modules given by Xilinx cannot be modified in any way because their VHDL source code is not provided. Also, the proposed generic

algorithm only generates pipelined multipliers with a latency included between 0 and $\lfloor log_2(n-2) \rfloor + 1$ inclusively. However, in Table 2.3, if we compare reference multipliers with those generated by the generic algorithm for similar latencies, one can see that our multipliers are about 10% faster than the ones included in the Xilinx library. Otherwise, the circuits given by Xilinx are substantially smaller with generally 35% fewer slices than our generic multiplier, and can be up to 50% smaller for the 8x8 multipliers with one pipeline stage. We should however consider the fact that the designs coming from Xilinx's library are placed and routed especially for Virtex devices. Also, such modules are often optimized manually to improve their density over designs produced with automated tools.

To have a better idea of the frequency and complexity variation depending on the length of the multipliers generated, we compiled results of Table 2.3 to produce different graphs. The first one, depicted in Figure 2.4, shows the behaviour of the maximum reachable frequency for different lengths. The second graph in Figure 2.5 illustrates the variation of the complexity, expressed in number of slices, depending on the length of the multipliers. Both figures show two curves; one for combinational multipliers and the other for completely pipelined multipliers.

Figure 2.4 clearly illustrates that the maximum frequency attainable for a pipelined multiplier is always higher than the one for a combinational multiplier of the same length. Also, in Figure 2.5, the number of slices needed by a pipelined multiplier is always

superior to the number of slices required by a combinational multiplier of the same length. As expected, pipelined multipliers are faster than combinational ones, but are also more complex. By being at the extremes of all the possible pipeline configurations, with registers everywhere for one case and no register at all for the other case, these curves provide intervals inside which the frequencies and the complexities for all the other pipeline configurations should remain.

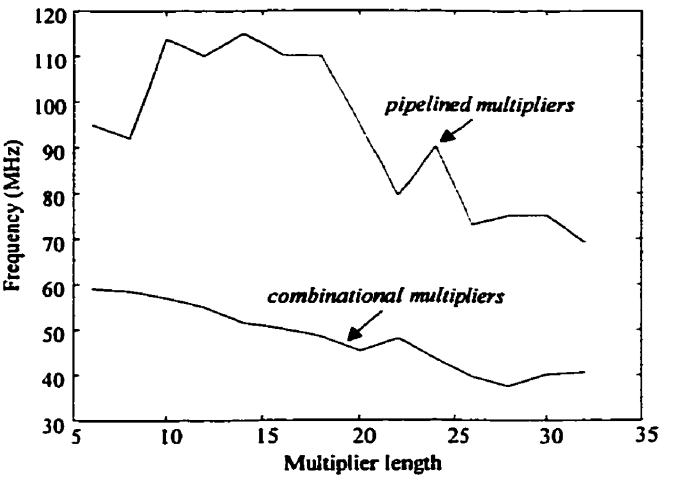

Figure 2.4 Variation of the maximum frequency for pipelined and combinational multipliers depending on multiplier length

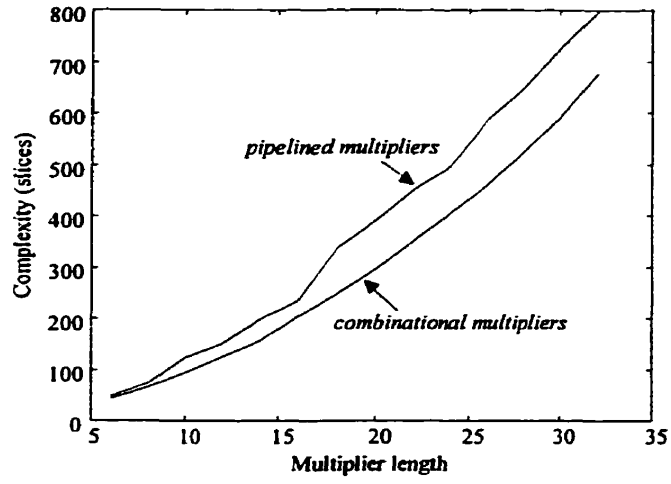

Figure 2.5 Variation of complexity for pipelined and combinational multipliers depending on multiplier length

As expected, Figure 2.5 shows clearly that complexity increases with the length of the multipliers generated by the proposed algorithm. By using data from Table 2.3, we can find the polynomial expansion of the circuit complexity with the increase of multiplier length for fully pipelined (equation (2.11)) and combinational (equation (2.12)) multipliers. The following expressions were developed using the "polyfit" function in Matlab, which finds the coefficients of a polynomial that fits the data in a least square sense.

$$\text{Number of slices (pipelined)} = 0.4394n^2 + 12.8397n - 55.6772 \qquad (2.11)$$

$$\text{Number of slices (combinational)} = 0.5118n^2 + 4.7036n - 4.5591 \qquad (2.12)$$

Otherwise, the behaviour of the maximum reachable frequency is less predictable than the complexity. One would have expected, in Figure 2.4, the processing speed to monotonically decrease with the increase of complexity, which is not the case for every lengths. It is important to specify that the timing analysis is executed only once the circuit has been placed and routed, unlike the area complexity that is determined before. Therefore, the timing analysis tool included in Alliance 2.1i includes the interconnection delays in its frequency estimations. Since Xilinx states that a typical design should have about 60% of its delays caused by interconnections in a Virtex FPGA, the results of the place and route process have a great impact on the maximum reachable frequency. Also, the frequencies given by the timing analyser included in Alliance are only estimations based on a sampling of all the possible paths that introduce delays. This value can also vary once the circuit is implemented in the FPGA. That can explain part of the irregularity in the speed versus size inverse relation. Nevertheless, for pipelined and combinational multipliers, the maximum operation frequency tends to decrease when the multiplier length expands. These relations are expressed in equation (2.13), for pipelined multipliers, and equation (2.14) for combinational ones. Given the irregular behaviour of our results, the reader should keep in mind that both equations (2.13) and (2.14) are only estimations, and they should not be used to evaluate precisely the maximum frequency of a given multiplier.

$$\text{Maximum frequency (pipelined)} = -0.1159n^2 + 2.9139n + 87.1418 \qquad (2.13)$$

$$\text{Maximum frequency (combinational)} = 0.0123n^2 - 1.3167n + 67.9595 \quad (2.14)$$

2.6 Conclusion

Several techniques (Cavanagh, 1984) exist to perform high-speed multiplication, like Booth algorithm, array multiplication, table lookup multiplication and Wallace trees. Each one has advantages and drawbacks. However, without being the most optimised or the most compact available, the algorithm explained in this paper is very flexible. Also, it offers a good compromise between speed and area that may be useful with many applications. The proposed multiplier generator algorithm can be implemented in VHDL or in any other hardware description language that integrates automated module generation controlled by conditional and loop statements.

A reconfigurable multiplier like the one presented in this paper is very useful if not indispensable in the context of customizable processors. It can be adapted to almost every length by simply modifying a parameter. An equally interesting characteristic of this multiplier is the possibility to adjust its latency to insert it in any architecture that has a fixed pipeline depth. Furthermore, a designer also has the flexibility to configure the way pipelining is realized inside the multiplier to more closely match the delays of the other modules surrounding it. A VHDL code based on the generic algorithm can be fully compatible with the other components and addresses all the multiplication lengths and pipeline depth by modifying only few parameters. Every new multiplier is then

synthesized without the need to validate it every time an architectural change is made, since it is already validated for all lengths and latencies.

By contrast, a library based on custom implementations may provide comparable speed designs, with fewer resources, but without the same level of flexibility. It does not allow the possibility to work with all multiplier lengths and latencies. A library also restricts a designer to choose between a limited selection of multipliers and replacing the existing module with the new one for each different synthesized processor. Also, a VHDL code is more flexible because it can be entirely integrated in an existing module hierarchy composing a complex processor. A unique code allows more control and better integration by grouping all the synthesis steps together, instead of having to use different tools to generate all the modules. Finally, it is totally portable to other target technologies like other FPGA brands or ASICs.

CHAPITRE 3

ARCHITECTURE INITIALE ET TECHNOLOGIE VISÉE

Ce chapitre présente de manière générale le processeur de traitement de signaux PULSE, qui a servi de point de départ au travail relaté dans ce mémoire. Une importance particulière est accordée au chemin de données, étant donné qu'il constitue le cœur de ce mémoire. Sans entrer dans les détails, une vue globale de son architecture est nécessaire, afin de bien comprendre le rôle de chaque module, ainsi que son interaction avec les autres blocs fonctionnels. De plus, à la lumière des explications qui suivent, les choix de conception et les modifications apportées au modèle initial prendront tout leur sens. Suite à la présentation de l'architecture PULSE, une section est dédiée à la technologie visée pour l'implantation du processeur. Toutes les particularités des FPGA de la famille Virtex de Xilinx qui sont pertinentes à l'utilisation que nous comptons en faire seront exposées. En fait, ce choix de FPGA s'impose de lui-même lorsqu'on examine de près certaines de ses caractéristiques. C'est d'ailleurs ce que cette section a pour but de démontrer. Finalement, la troisième et dernière partie de ce chapitre conclura en présentant les divers outils logiciels utilisés tout au long du développement et de la validation du chemin de données.

3.1 Architecture PULSE

Le processeur DSP PULSE fut initialement développé pour une implantation dans un ASIC, dans l'optique d'effectuer du traitement de signaux vidéos de qualité professionnelle. Cette application nécessite des performances de pointe ainsi qu'une capacité de calcul en temps réel. PULSE est aussi adapté à tous les domaines pouvant bénéficier de traitement rapide de données entières ou à virgule fixe, car la possibilité de travailler en virgule flottante n'est pas implantée.

3.1.1 Architecture de haut niveau

L'architecture est composée principalement d'un contrôleur et d'un chemin de données hautement parallèle. En effet, celui-ci comporte plusieurs PE identiques où tous les calculs sont réalisés. Le chemin de données est de type SIMD, ce qui signifie que la même instruction est envoyée à tous les PE qui l'exécutent chacun sur différentes données. Le contrôleur a donc le rôle primordial de distribuer et de synchroniser les données entrant et sortant des PE, de telle sorte qu'elles puissent être disponibles au bon moment et que l'utilisation de tous les PE soit optimale. Ceci est réalisé avec l'aide des canaux de communication nord et sud illustrés à la Figure 3.1. Ces canaux permettent différentes configurations de transfert de données à l'intérieur du chemin de données, selon l'algorithme à effectuer. L'architecture inclut aussi deux générateurs d'adresses permettant l'accès non linéaire aux données en mémoires externes, selon des patrons entrés sous forme de paramètres. L'exécution de certains algorithmes comme la FFT est

grandement améliorée et accélérée grâce aux générateurs d'adresses. On retrouve finalement une interface JTAG utile aux tests d'intégrité des circuits une fois ceux-ci implantés sous forme de ASIC. Mentionnons toutefois que cette caractéristique a été supprimée, étant donnée l'implantation dans un FPGA qui possède déjà une telle interface.

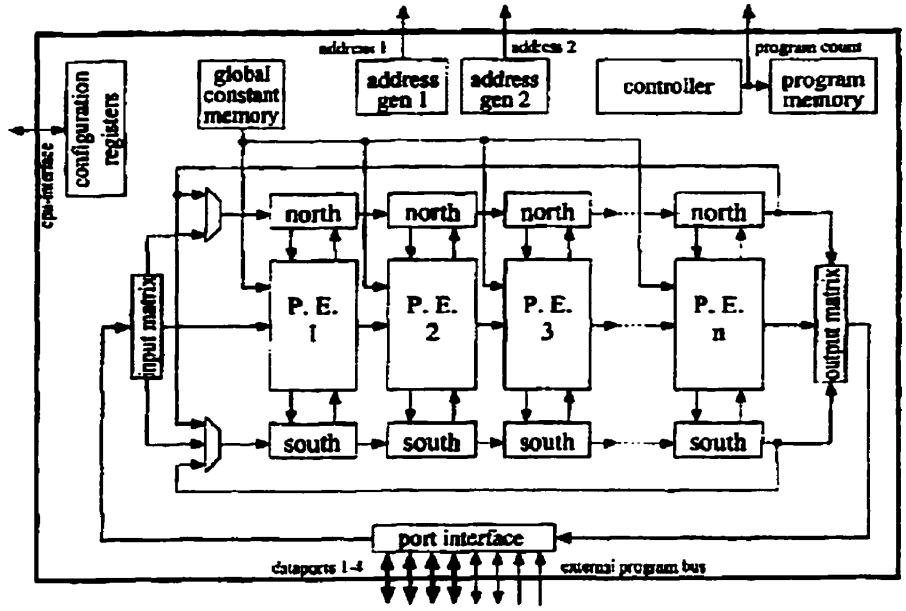

Figure 3.1 Canaux de communication (*Marriott et al., 1998*)

3.1.2 Architecture du chemin de données

Le chemin de données initial est composé de quatre PE effectuant des opérations sur des données de 16 bits exprimées en complément à deux. Ceux-ci constituent le cœur du processeur, car ils comprennent différents modules de calcul qui leur permettent

d'effectuer une grande variété d'opérations possibles sur les données. L'architecture interne d'un PE, montrée à la Figure 3.2, comporte 4 bus source et 4 bus destination, servant respectivement à distribuer les données entrant dans les différents modules et à diriger les données sortantes vers les sorties du PE. Afin de séquencer les différentes étapes de traitement et d'accélérer la fréquence maximale d'opération, un pipeline de 4 étages est employé. La première phase est la lecture des données à traiter, la deuxième phase transmet ces données aux bus sources et commence le traitement, la troisième phase termine le traitement et la quatrième phase écrit les résultats sur les bus destination qui redirigent ensuite celles-ci vers les sorties ou d'autres structures internes. Mentionnons aussi que chaque PE peut être individuellement activé ou désactivé par le contrôleur pour permettre une plus grande flexibilité de fonctionnement.

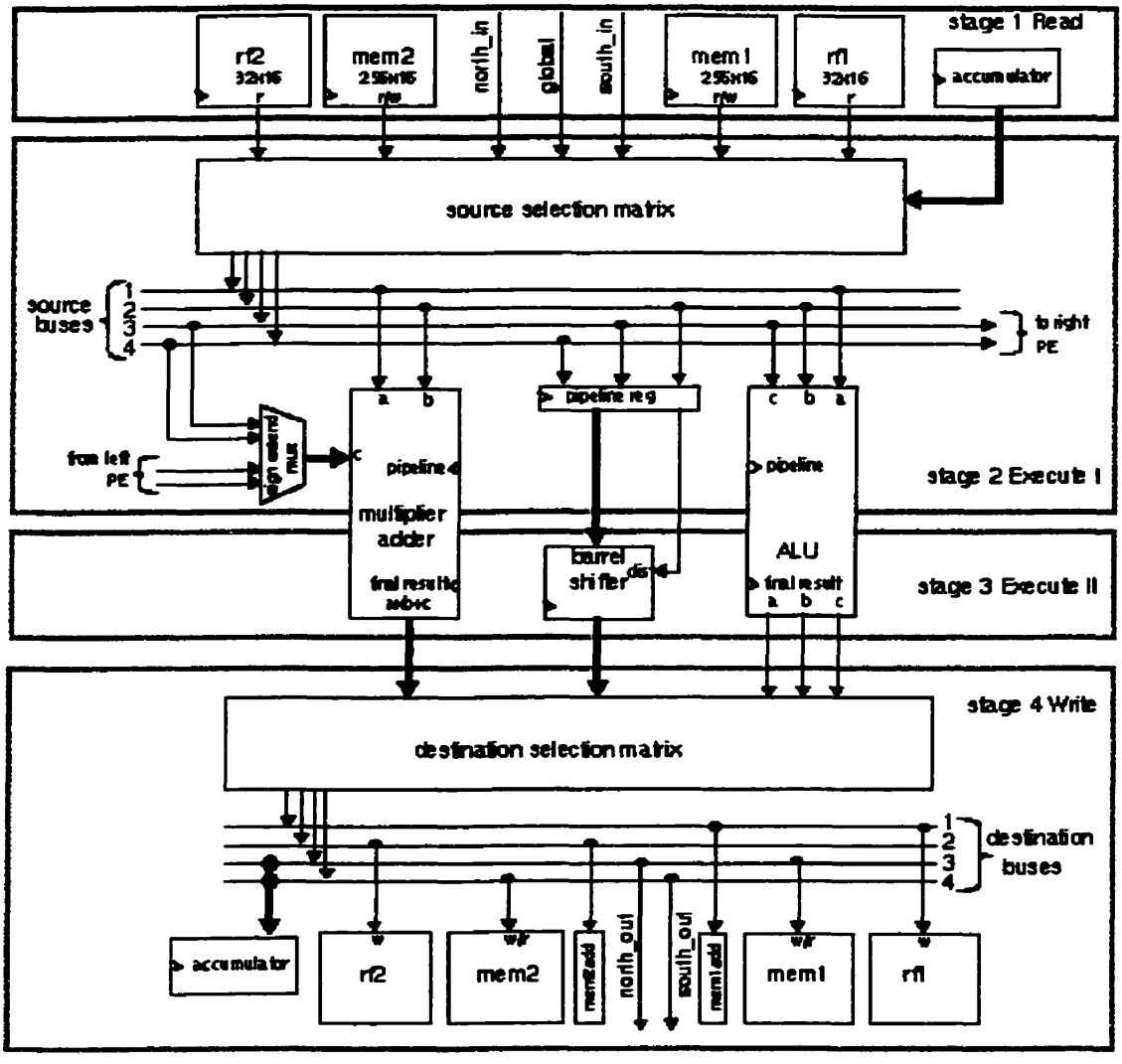

Figure 3.2 Architecture interne initiale des PE (*Marriott et al., 1998*)

Chaque module joue un rôle bien précis à l'intérieur de l'architecture des PE et les lignes qui suivent expliquent succinctement la fonctionnalité de chacun d'eux ainsi que les particularités qui y sont rattachées.

- Décaleur barillet (32 bits) : Effectue le décalage ou la rotation des bits du mot à traiter vers la gauche ou la droite, selon la distance demandée. En plus du mode de fonctionnement logique conventionnel, ce décaleur barillet peut fonctionner en mode arithmétique pour conserver le signe du mot à traiter.

- Multiplicateur-sommateur (16x16+32 bits) : Multiplie deux termes, pour ensuite en additionner un troisième au besoin. Le terme à additionner peut être lu des bus source, mais il peut aussi provenir du PE précédent, ce qui crée une chaîne d'accumulation entre les PE pour accélérer l'exécution de certains algorithmes.

- ALU (16 bits) : Effectue toutes les opérations logiques ainsi que les opérations arithmétiques d'addition et de soustraction. Différentes fonctions de comparaison et de permutation des entrées sont aussi disponibles. Quatre signaux donnent l'état du résultat de l'opération en cours (nul, négatif, débordement et retenue).

- Accumulateur (32 bits) : Accumule ou retranche selon le signe de la valeur d'entrée. Il peut être initialisé à une valeur donnée. Sa particularité réside en sa précision interne de 33 bits, lui permettant de détecter les débordements sur 32 bits, qu'ils soient signés ou non, et de continuer l'accumulation jusqu'à ce que la valeur interne de 33 bits retombe sous le seuil de débordement sur 32 bits. Le comportement de la sortie en présence de débordements peut plafonner à la valeur maximale permise, ou encore afficher les 32 bits les moins significatifs de la valeur interne.

- Registres (16 bits) : Au nombre de deux, ils ont une capacité de 32 mots chacun et la lecture et l'écriture de données peuvent se faire simultanément grâce à deux ports indépendants.

- Mémoires (16 bits) : Elles sont au nombre de deux et ont une capacité de 256 mots chacune. Contrairement aux registres, les opérations de lecture et d'écriture ne peuvent se faire en même temps car un seul port bidirectionnel pour les données est disponible. Les mémoires supportent l'adressage direct et indirect.

- Pile d'activité : Bien qu'absente de la Figure 3.2, car elle ne sert pas aux calculs, la pile d'activité joue le rôle de contrôleur interne aux PE, en effectuant les opérations conditionnelles locales afin de n'activer le PE que pendant le branchement conditionnel requis. Cette fonctionnalité permet l'utilisation d'instructions conditionnelles (si... alors... sinon...) dans les algorithmes.

3.2 Technologie d'implantation

Les circuits intégrés dédiés se sont graduellement imposés comme la plate-forme de choix pour l'implantation de processeurs ou de tout autre design d'envergure de haute performance. Lorsqu'un circuit intégré est conçu pour répondre aux besoins d'une application spécifique, on parle de ASIC (*Application-Specific Integrated Circuit*). Les

ASIC permettent un niveau d'intégration supérieur aux autres options disponibles pour mettre en œuvre du matériel dédié et ils permettent des fréquences d'horloge plus rapides, au détriment de coûts initiaux élevés pour la fabrication des masques. Ils ont cependant le désavantage d'être figés une fois réalisés. Cependant, les FPGAs ont récemment fait des pas de géant et les compagnies qui les commercialisent sont en mesure d'offrir présentement sur le marché des circuits programmables dont le degré d'intégration et les fréquences d'opération offrent la possibilité de supporter des designs relativement complexes et performants.

Un des fabriquants de FPGA les plus importants, Xilinx, a mis sur le marché une famille de circuits destinés aux applications lourdes. Celle-ci, la famille Virtex (Virtex product specifications. Xilinx, Inc.), possède plusieurs caractéristiques intéressantes et adaptées entre autres, à l'implantation de processeurs DSP, comme celui dont il est question dans ce mémoire. L'utilisation de FPGA plutôt que de ASIC comme support pour notre processeur diminue substantiellement le temps de développement et facilite la phase de déverminage, sans compter la diminution des coûts à petite échelle. La présente section met en évidence certaines particularités et spécifications de la famille Virtex, en vue de l'utilisation comme plate-forme pour notre processeur configurable

Le modèle sélectionné pour une première implantation du processeur est un circuit qui nous est accessible et qui offre un bon rapport performance-prix, le Virtex XCV800 avec un indice de vitesse de 5. Il possède plus de 880 000 portes logiques équivalentes, sans

compter les quelques 115 000 bits de mémoire disponibles dans un ensemble de mémoires encastrées appelées *BlockRAM*. Sa fréquence d'opération maximale recommandée est de 180 MHz. Le nombre maximal d'entrées/sorties utiles est de 512 et il comporte une interface de test « boundary scan » JTAG IEEE 1149.1. Comme tous les FPGA de Xilinx, le XCV800 est reprogrammable à volonté, étant donné que sa configuration est emmagasinée dans des mémoires statiques (SRAM). Toutes ces caractéristiques sont obtenues grâce à un procédé de fabrication CMOS de 0,22 micron à 5 couches de métal superposées.

3.2.1 Mémoire disponible

Deux types de mémoire sont accessibles à l'intérieur des FPGA de la famille Virtex et les différents paramètres de celles-ci sont modifiables, ce qui offre un degré de flexibilité avantageux. Premièrement, la *SelectRAM* est constituée de CLB (*Configurable Logic Block*) qui sont utilisés pour implanter le circuit programmé dans le FPGA. La SelectRAM exploite la possibilité d'utiliser les CLB comme mémoire. 301 056 bits de mémoire sont disponibles sous cette forme. Cette stratégie offre donc de la mémoire très rapide directement incorporée dans le circuit. Elle diminue cependant le nombre de CLB disponibles pour l'implantation du circuit fonctionnel. Deuxièmement, les BlockRAM sont des blocs de mémoire configurables, regroupés en deux banques placées de part et d'autre de la matrice de CLB, tel qu'illustré à la Figure 3.3. Ces BlockRAM, au nombre de 28 dans un XCV800, permettent l'utilisation d'un ou deux ports et ont chacun une

capacité de 4096 bits pouvant être répartis en mots de longueur variable de 1, 2, 4, 8 ou 16 bits. L'avantage des BlockRAM est que, contrairement à la SelectRAM, ils n'utilisent pas de ressources destinées au reste du circuit à implanter.

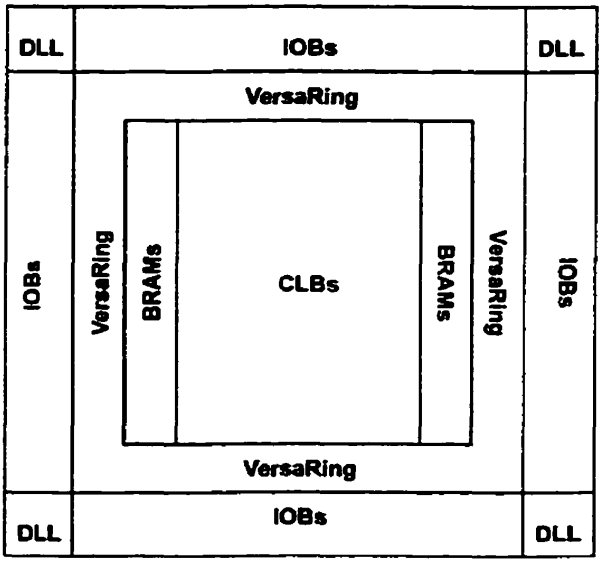

Figure 3.3 Disposition des BlockRAM et de l'anneau VersaRing (*Virtex product specifications. Xilinx, Inc.*)

3.2.2 Interconnexions

Afin de faciliter l'étape du placement et routage du circuit à implanter, Xilinx incorpore dans ses FPGA Virtex des blocs de routage (*VersaBlock*) et un anneau de routage (*VersaRing*). L'anneau est montré sur la Figure 3.3et les VersaBlock sont distribués dans toutes la matrice de CLB pour interconnecter localement différentes ressources entre elles. De façon similaire, le VersaRing optimise les entrées/sorties en procurant des

ressources additionnelles de routage en périphérie du circuit. Ces deux structures deviennent particulièrement importantes lorsqu'on sait que le délai imputable aux interconnexions internes représente plus de 60% du délai total, ce qui signifie que les opérations logiques sont plus rapides que les délais de propagation dans un FPGA Virtex.

3.2.3 Logique arithmétique

Un des aspects les plus importants qui guide notre choix vers la famille Virtex est la présence de ressources logiques dédiées aux opérations arithmétiques. De manière à accélérer les additions sur plusieurs bits, deux chaînes de propagation de retenue rapide ont été implantées à l'intérieur de chaque CLB. Toujours pour l'addition, plusieurs portes OU exclusif sont présentes dans les CLB. De plus, des portes ET dédiées sont disponibles afin de permettre une implantation efficace de multiplicateurs. Ces particularités rendent l'exécution de fonctions arithmétiques plus efficace sur ce FPGA et permettent d'atteindre des performances intéressantes comme le montre le Tableau 3.1.

Tableau 3.1 Délais d'exécution d'opérations arithmétiques

Function	Bits	Virtex -6
Register-to-Register		
Adder	16 64	5.0 ns 7.2 ns
Pipelined Multiplier	8 x 8 16 x 16	5.1 ns 6.0 ns
Address Decoder	16 64	4.4 ns 6.4 ns
16:1 Multiplexer		5.4 ns
Parity Tree	9 18 36	4.1 ns 5.0 ns 6.9 ns
Chip-to-Chip		
HSTL Class IV		200 MHz
LVTTL,16mA, fast slew		180 MHz

3.2.4 Tampons à trois états

On peut finalement mentionner que chaque CLB comporte 2 tampons à trois états, ce qui est très utile pour la réalisation de bus partagés par plusieurs modules qui doivent y écrire et lire des données. D'ailleurs, les bus source et destination décrits dans la section précédente pourront profiter de cette ressource.

3.3 Outils logiciels

Comme dans tout travail de développement d'envergure, plusieurs outils ont été employés à différents stades du développement. La plate-forme de travail étant le PC, tous les outils ont donc été utilisés dans leur version PC. Cette section décrit chaque outil logiciel en précisant quelques particularités relatives à leur usage. Aussi, les limitations

de ceux-ci, de même que les problèmes qu'elles ont occasionnées et les solutions de rechange sont abordées. On peut classifier les outils utilisés pour trois grandes catégories, soit la simulation ou validation du code VHDL, sa synthèse et finalement, le placement-routage. Bien qu'elle n'ait pas été réalisée dans le cadre de ce travail, l'étape subséquente serait de simuler la fonctionnalité une seconde fois en incorporant les délais occasionnés par l'étape du placement-routage au modèle. Cette procédure permettrait alors de détecter les problèmes de synchronisme potentiels. Cependant, l'implantation dans un FPGA reprogrammable nous permet de sauter cette étape et de procéder aux tests fonctionnels directement sur le circuit, pour ensuite y apporter les correctifs nécessaires en reprogrammant le FPGA.

3.3.1 Simulation et validation

La simulation est indispensable pour s'assurer du bon fonctionnement du circuit. En simulant toutes les fonctions attendues, le concepteur peut donc valider son modèle et ainsi confirmer qu'il respecte les spécifications demandées. Deux outils ont été utilisés pour effectuer les différentes simulations requises pour chaque module. Le premier, ModelSim de la compagnie ModelTech (HDL Simulation Products. Model Technology, Inc.) a été employé uniquement pour les simulations fonctionnelles de haut niveau du modèle complet du processeur. Toutefois, pour les simulations de tous les modules du chemin de données de façon individuelle, l'outil Active-VHDL version 3.3, distribué par la compagnie Aldec (Active-HDL Standard Edition Datasheet. Aldec, Inc.), a été utilisé.

En fait, son interface de développement intuitive permet une organisation hiérarchique des fichiers. Sa facilité d'utilisation et les différentes options disponibles pour faciliter les tâches de simulation, comme la génération automatique de bancs de test par exemple, en font un outil puissant et versatile. C'est d'ailleurs pourquoi la presque totalité des simulations ont été réalisées avec cet outil plutôt que ModelSim.

3.3.2 Synthèse

La synthèse constitue la seconde étape de développement, une fois le code VHDL complété et validé. Cette étape permet de produire un circuit logique correspondant à la description VHDL analysée et compilée. Pour cette phase de développement, nous utilisons l'outil Synplify de la compagnie Synplicity (Product Literature. Synplify, Inc.). Il est relativement simple à utiliser et permet une synthèse rapide et efficace du code VHDL, optimisée pour l'architecture de FPGA de la technologie cible. Synplify compile le code selon le circuit ciblé, pour obtenir une meilleure estimation des délais et des ressources utilisées. Cette analyse permet donc d'obtenir un estimé de la fréquence maximale atteinte et des ressources utilisées. Toutefois, l'expérience a montré que ces valeurs approximatives sont peu fiables et que les résultats correspondants obtenus par l'outil de placement-routage sont parfois très différents, ce qui fait que cette fonctionnalité n'a pas été utilisée.

Une particularité très intéressante de Synplify est son utilitaire HDL Analyst qui, une fois le code VHDL compilé, permet d'avoir une vue schématique du circuit généré au niveau des registres. Chaque module y est montré ainsi que tous les signaux qui s'y connectent. De plus, plusieurs niveaux hiérarchiques y sont représentés, ce qui permet une vue globale des modules ou une vue plus en profondeur d'un module en particulier. Ce schéma identifie d'un trait rouge le chemin critique pour une meilleure compréhension des limites de performance du circuit étudié.

3.3.3 Placement et routage

L'étape suivant la synthèse est le placement et le routage des ressources qui s'effectuent avec le logiciel Alliance de Xilinx (Alliance Series v2.1I Software. Xilinx, Inc.). La synthèse ayant généré le circuit équivalent du code VHDL d'entrée, il faut maintenant répartir ce circuit dans le FPGA visé, de manière à limiter les délais de propagation et obtenir une densité optimale du circuit. Pour ce faire, deux étapes sont nécessaires. La première, le placement, distribue toutes les composantes de bas niveau du circuit synthétisé (portes logiques, registres, multiplexeurs, etc...) dans le FPGA, pour en exploiter les ressources disponibles de façon optimale, c'est-à-dire en obtenant un circuit compact et rapide. La seconde étape, le routage, a pour but d'interconnecter tous ces modules placés dans le FPGA, pour obtenir des communications efficaces et rapides entre les composantes du circuit.

Afin d'automatiser ce processus, une macro a été créée, dans laquelle chaque ligne de commande appelle un sous-programme d'Alliance qui doit être exécuté. Il faut cependant mentionner que pour chacune des commandes à effectuer, plusieurs options et paramètres sont disponibles afin de guider l'outil pour une meilleure optimisation. Voici donc ces commandes avec les options utilisées.

ngdbuild -p xcv800-5-bg432 %1.edf %1.ngd

Utilisée simplement pour convertir le fichier d'entrée généré par Synplify en un fichier pouvant être lu et interprété par Alliance. La seule option disponible à ce point est le modèle de FPGA ciblé.

map -p xcv800-5-bg432 -cm speed -o %1.ncd %1.ngd %1.pcf

La seconde ligne du fichier de commandes sert à porter le circuit dans le modèle de FPGA spécifié. Ceci correspond à trouver les équivalences entre les différentes composantes du circuit synthétisé par Synplify et les ressources disponibles dans les CLB du FPGA. La seule option disponible (-cm) sert à définir un mode de compilation privilégiant la vitesse ou la surface.

par -dfs -w -ol 5 %1.ncd %1.ncd %1.pcf

La troisième commande est la plus critique car elle démarre le placement et le routage. Elle doit répartir tous les circuits générés à l'étape précédente dans le FPGA, pour obtenir des interconnexions rapides et compactes entre les modules. Cette étape est déterminante

quant à la fréquence maximale atteinte et à la surface de silicium utilisée. Les paramètres servent à définir une analyse temporelle rapide ou plus poussée (-dfs) des chemins critiques, et le niveau d'effort (-ol, compris entre 1 et 5), qui correspond au compromis entre le temps de calcul et le résultat final.

trce -dfs %1.ncd %1.pcf -e 3 -o %1.twr

L'étape finale sert simplement à analyser les résultats du placement et du routage pour évaluer les performances temporelles de tous les chemins possibles. Cette analyse permet donc d'identifier les chemins critiques pour que le concepteur puisse ensuite modifier le circuit en conséquence. La seule option intéressante permet de choisir l'un des deux types d'analyse (-dfs), comme pour le placement-routage. Un problème rencontré avec cet outil vient du fait que la fréquence cible spécifiée à l'outil Synplify lors de la synthèse affecte grandement l'estimé de la fréquence maximale. Pour contrer cet inconvénient, plusieurs itérations de placement et de routage doivent être effectuées avec différentes fréquences cibles pour Synplify. Cette procédure, bien qu'elle soit très longue, permet néanmoins d'obtenir le circuit le plus rapide parmi tous ceux générés.

Finalement, rappelons que dans des FPGA complexes comme les Virtex, il est spécifié que plus de 60% des délais rencontrés sont imputables aux délais dans les interconnexions. Il est donc capital de bien maîtriser toutes les options disponibles, qui peuvent apporter des gains de performance notables, mais aussi dégrader substantiellement les performances si elles sont mal exploitées.

CHAPITRE 4

OPTIMISATION ET PARAMÉTRISATION

Ce dernier chapitre expose le travail réalisé sur l'architecture de départ du processeur PULSE. Notre objectif consistait à optimiser, paramétrer et rendre modulaire le chemin de données conçu initialement pour des mots de 16 bits. En fait, le développement de PULSE ayant été réalisé pour une implémentation dans un ASIC, plusieurs ajustements ont dû être apportés au modèle afin de migrer vers une réalisation du circuit dans un FPGA de la série Virtex de Xilinx. Une fois ces modifications terminées, la tâche suivante devait rendre le code VHDL du chemin de donnéesconfigurable à plusieurs niveaux; non seulement, ce code doit-il être paramétrable (configuration n'affectant pas l'architecture) mais il doit aussi être modulaire (configuration affectant l'architecture), c'est-à-dire être synthétisé en entier ou en parties lorsque l'application demandée ne requiert pas toutes les fonctionnalités disponibles. De façon pratique, les différents blocs ou modules constituant le chemin de données peuvent être inclus ou non dans le circuit final du FPGA.

L'idée derrière cette modularité est de disposer éventuellement d'un outil d'analyse automatique du programme de l'application qui permettra d'identifier les modules nécessaires à son exécution. Un fichier VHDL sera alors généré pour indiquer à l'outil de synthèse Synplify quels sont les blocs fonctionnels à conserver et ceux ne faisant pas partie du circuit. En procédant de la sorte, le chemin de données pourra être adapté

automatiquement à la tâche à réaliser et ainsi être plus compact que si tous les modules étaient conservés. De plus, l'architecture du processeur PULSE étant de type SIMD, le nombre de PE compris dans le chemin de données peut être aussi modifié par un simple paramètre lors de la synthèse. Ceci implique que plus la surface occupée par un PE est petite, plus le nombre de PE dans le chemin de données peut être augmenté, ce qui a pour conséquence de hausser substantiellement le nombre d'exécutions réalisées en parallèle et ainsi accroît la capacité de traitement du processeur.

Trois grandes parties composent ce chapitre. Premièrement, les modifications effectuées sur le code VHDL initial permettant une meilleure intégration dans le FPGA ciblé sont expliquées. D'autres améliorations ont aussi été apportées à la structure afin d'offrir plus de flexibilité pour la programmation des applications et pour y ajouter de nouvelles fonctionnalités. Deuxièmement, tout l'aspect paramétrisation et modularité est traité en exposant les nouvelles possibilités et les différentes configurations possibles du chemin de données. Finalement, des mesures de complexité sont fournies pour des fins de comparaison entre le modèle initial et la nouvelle architecture flexible.

4.1 Optimisation et modification de l'architecture

Dans cette section, nous allons traiter des différentes étapes franchies pour la migration vers une implémentation dans un FPGA Virtex en adaptant l'architecture pour tirer parti des fonctions qui y sont intégrées. Pour ce faire, la présente section explique les

modifications qui ont dues être apportées à différents modules. Elles ont pour objectif d'exploiter plus efficacement les ressources disponibles à l'intérieur du FPGA et d'améliorer l'architecture de certains modules afin de permettre une fréquence de fonctionnement plus élevée ou un circuit synthétisé plus compact.

4.1.1 Modification des sélecteurs de bus

La présence de bus de source et de destination communs à plusieurs modules nécessite l'emploi de sélecteurs de bus qui servent à ne connecter qu'un module à la fois sur un bus afin d'éviter les conflits d'écriture sur ce bus. Dans la version originale du chemin de données pour une implémentation ASIC, les sélecteurs de bus illustrés à la Figure 4.1 étaient construits avec des multiplexeurs de 16 bits, à deux entrées et une sortie. Ainsi, pour les modules transmettant des valeurs sur les bus, chaque sortie entrait dans un multiplexeur dont l'autre entrée était fixe à zéro. La sélection d'un module pour l'écriture sur un bus se faisait en laissant passer la valeur vers la sortie du multiplexeur. Dans le cas contraire où le module n'était pas sélectionné, l'entrée à zéro était transmise à la sortie du multiplexeur. Ensuite, les sorties de tous les multiplexeurs communs à un bus étaient envoyées aux entrées d'une porte logique OU à 2, 3 ou 4 entrées de 16 bits pour déterminer la donnée à inscrire sur le bus. Une telle porte devait être connectée à chacun des huit bus de source et de sortie. Cette stratégie fut employée par l'équipe de développement, pour contourner certains problèmes d'utilisation des tampons à trois

états. En effet, cette structure n'était pas supportée par la fonderie qui devait produire les processeurs PULSE.

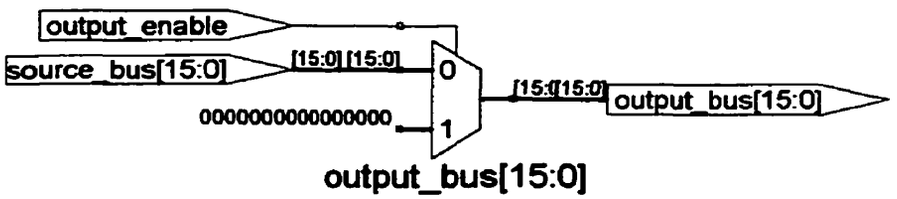

Figure 4.1 Multiplexeur d'un sélecteur de bus de la version initiale du chemin de données

Dans un Virtex, chaque CLB possède un tampon à trois états. Ceux-ci peuvent remplacer directement la structure expliquée précédemment en connectant ou déconnectant les sorties des modules aux différents bus vers lesquelles elles sont destinées. On peut donc, en recourant aux tampons à trois états présents dans le circuit, supprimer les multiplexeurs et les portes OU, tout en conservant la même fonctionnalité. Ceci a permis de libérer plus de 125 CLB, (ce qui représente environ 19% des ressources initiales du chemin de données. La nouvelle structure des sélecteurs de bus est illustrée à la Figure 4.2.

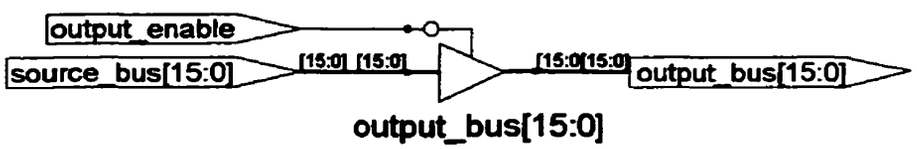

Figure 4.2 Sélecteur de bus utilisant les tampons à trois états des FPGA Virtex

4.1.2 Modification des mémoires et registres

Tel que présenté dans la section sur la technologie Virtex dans le chapitre précédent, les FPGA de cette famille disposent de blocs de mémoire configurables nommée BlockRAM. L'implantation de mémoires ou de registres peut donc se faire par cette alternative, ou encore en recourant à la SelectRAM qui nécessite cependant l'utilisation de CLB et diminue donc la quantité de ressources disponibles pour le reste du circuit. Le compromis réalisé consiste à utiliser les BlockRAMs pour l'implantation des deux mémoires incluses dans chaque PE. Malgré le fait qu'elles soient configurables, les mémoires du chemin de données ont habituellement une capacité de 256 mots de 16 bits, ce qui correspond à la taille exacte d'un BlockRAM. Par ailleurs, l'utilisation de SelectRAM aurait nécessité 256 CLB par mémoire. On peut aussi mentionner que le FPGA XCV800 ciblé possède 28 BlockRAM. La syntaxe exigée par l'outil Synplify pour indiquer l'utilisation de BlockRAM lors de la synthèse du code VHDL est la suivante :

```
type Single_Port_Mem_type is array (0 to 256) of std_logic_vector(15 downto 0);
signal Mem_s : Single_Port_Mem_type;
attribute syn_ramstyle of Mem_s : signal is "block_ram";
```

Les registres, en ayant généralement une capacité moindre que les mémoires, sont présentement implantés à l'aide de la SelectRAM. Cependant, il serait très simple de les

transférer éventuellement dans les BlockRAM restants. L'architecture des registres a été complètement revue. En effet, les registres originaux utilisaient les ressources alors disponibles dans la technologie visée, soit un ASIC de ChipExpress. Le circuit résultant était constitué de bascules D sensibles aux niveaux (*latches*) qui ont dû être remplacées par des bascules D synchrones qui ont l'avantage d'être déjà incluses dans la structure interne du FPGA.

4.1.3 Optimisation du module additionneur-soustracteur

Ce module compris dans l'unité logique et arithmétique sert à effectuer une addition ou une soustraction de deux mots de 16 bits, A et B, représentés en complément à deux. Le bit d'entrée ADD_SUB sert à faire cette sélection. De plus, ce module doit être en mesure de réaliser ces deux opérations en utilisant une retenue d'entrée identifiée CI. Lorsque celle-ci vaut 1, on doit obtenir comme résultat A+B+1 ou A-B-1 selon l'opération choisie.

En examinant le résultat de la synthèse du code VHDL original, on constate que deux sommateurs sont requis et un multiplexeur fait la sélection de l'opération en sortie. Cette façon de faire implique qu'en tout temps, les deux opérations sont calculées alors qu'un seul résultat n'est requis à la fois. Ce circuit peut donc être simplifié pour diminuer la quantité de ressources utilisées tout en demeurant aussi performant. La Figure 4.3 illustre l'architecture du module original :

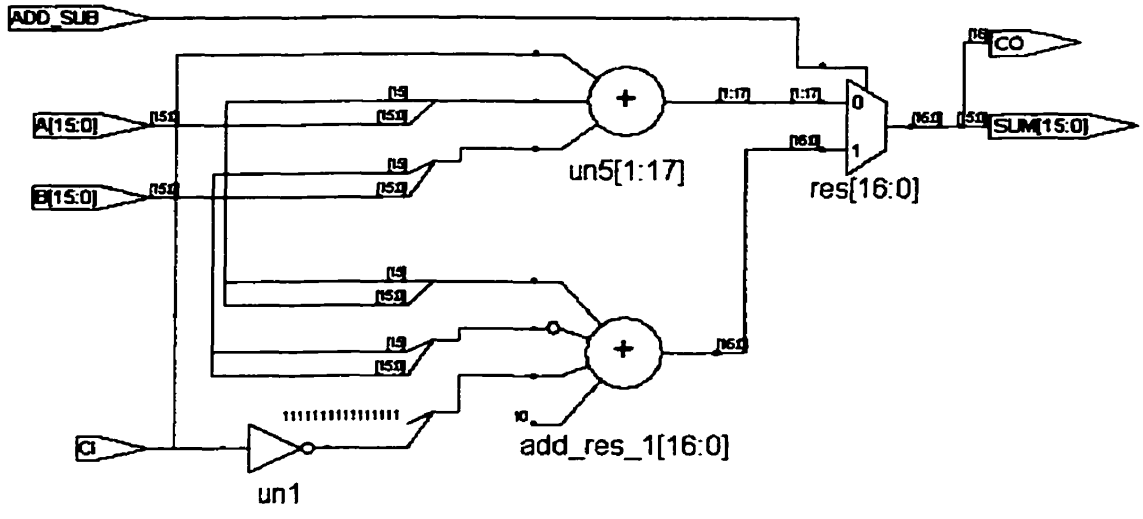

Figure 4.3 Architecture originale du module additionneur-soustracteur

La première idée exploitée pour améliorer ce circuit consistait à créer des descriptions complètes de toutes les parties du circuit pour pouvoir implanter une structure d'addition en *carry look-ahead*. Cependant, les résultats obtenus pour cette architecture ont montré une fréquence d'opération coupée de moitié et une surface d'implantation représentant presque trois fois celle du circuit original. Cette solution a donc été rejetée. Sa piètre performance et sa complexité sont dues au fait que les chaînes de propagation de retenues rapides ne sont pas utilisées. En effet, l'outil de synthèse Synplify ne reconnaît pas l'addition décrite et n'exploite donc pas ce signal optimisé. Pour ce faire, le code VHDL doit clairement comporter une addition de deux termes exprimée avec le signe « + ».

La solution finale fait appel aux simplifications pouvant être faites en tenant compte de certaines propriétés arithmétiques des nombres représentés en complément à deux. Une soustraction est en fait une addition du premier mot avec le second dont on a préalablement inversé tous les bits et auquel on a ajouté 1. Ainsi, cette stratégie exige simplement de faire la sélection de l'opération en premier lieu afin de modifier le mot B en conséquence et ensuite l'envoyer, avec le mot A, dans un seul sommateur. En ce qui concerne la retenue, elle est représentée sur le même nombre de bits que les termes A et B et ensuite additionnée avec ceux-ci après qu'elle eut été modifiée selon le choix de l'opération. Dans le cas d'une addition, la retenue vaudra 0...001 et pour une soustraction, elle vaudra 111...1 pour correspondre respectivement à 1 et −1 en représentation en complément à deux.

Le nouveau code VHDL modifié de l'additionneur-soustracteur fait donc appel à ces stratégies pour simplifier l'architecture et n'utiliser ainsi qu'un seul sommateur. Le multiplexage des valeurs en entrée a été remplacé par l'emploi de simples portes OU exclusifs qui réalisent en fait la fonction d'inverseurs commandés. L'architecture est illustrée à la Figure 4.4.

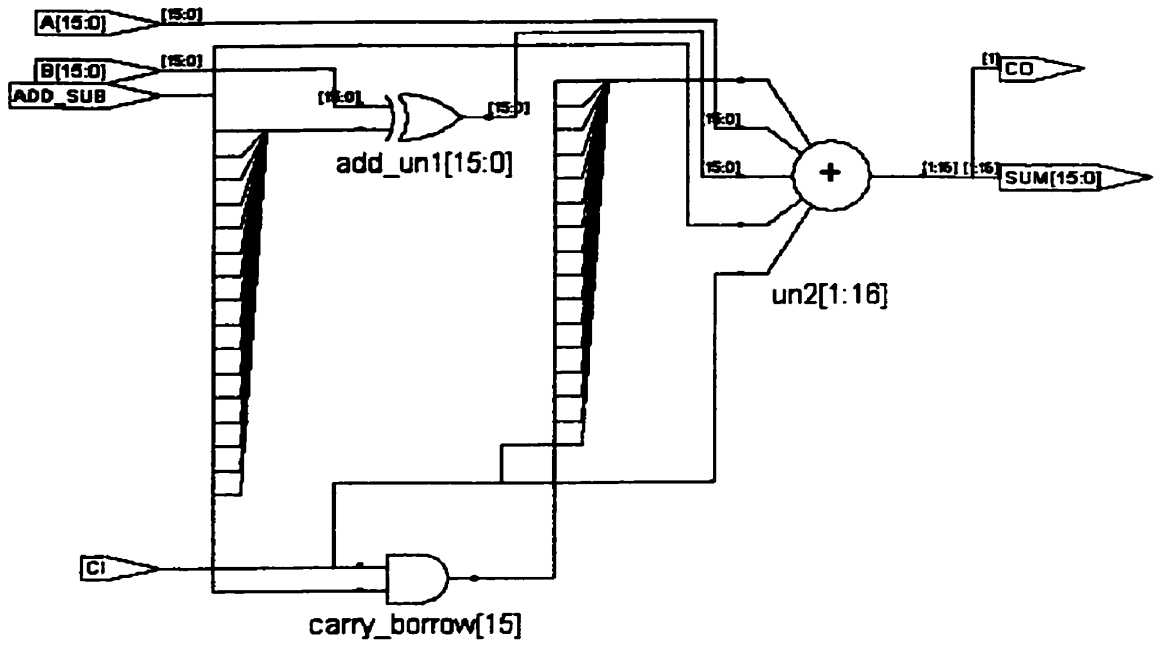

Figure 4.4 Architecture optimisée du module additionneur-soustracteur

La fréquence maximale d'opération ainsi que le nombre de CLB employés pour l'architecture initiale et l'architecture optimisée sont indiqués dans le Tableau 4.1. Ces deux mesures, estimées par l'outil de placement et routage Xilinx Alliance, sont représentatives de la vitesse de traitement et de la superficie qu'aura le circuit une fois implanté dans le FPGA. Comme on peut le constater, la fréquence d'opération augmente de 40% mais l'avantage se situe principalement au niveau de la surface d'implémentation requise qui ne représente plus que 32% de la surface initiale.

Tableau 4.1 Comparaison entre les architectures d'additionneur-soustracteur

Module	Fréquence (MHz)	Nombre de CLB
Original	49.3	14
Optimisé	68.8	5

4.1.4 ROM de décodage

Les communications à l'intérieur des PE sont réalisées par des bus source et destination auxquels sont connectés les différents modules. Afin de n'allouer l'écriture sur un bus que par un seul module à la fois, des signaux provenant du contrôleur sont nécessaires. Ceux-ci servent à gérer les autorisations d'écriture sur les bus pour chaque module afin d'éviter tout conflit pouvant survenir dans l'éventualité où plusieurs données seraient inscrites simultanément sur un même bus.

Pour minimiser le nombre de connexions voyageant du contrôleur vers les PE, les signaux sont encodés dans le contrôleur pour ensuite être décodés par une ou plusieurs ROM de décodage. En fait, le système de codage est le suivant : un signal est dédié à chacun des bus des PE et indique, à l'aide de 4 bits pour les bus de source et 3 bits pour les bus de destination, le module qui y est connecté pour l'écriture de données. Tous ces signaux entrent dans la ROM de décodage pour ensuite ressortir vers les PE sous forme de multiples signaux correspondant à chacune des sorties des modules. Ces signaux indiquent alors à la sortie contrôlée, le bus auquel elle doit de connecter. Pour ce faire,

chaque bit d'un tel signal représente un bus où 1 signifie une connexion et 0 une sortie flottante. Dans le cas où une sortie de module n'écrirait sur aucun bus, tous les bits du signal de contrôle seraient à zéro.

Cette ROM de décodage a été déplacée de sa position initiale de l'intérieur du contrôleur vers le chemin de données. En procédant ainsi, la longueur à parcourir par les signaux plus denses, une fois décodés, est moindre étant donnée la proximité de la ROM par rapport aux PE. Ainsi, le routage des interconnexions est simplifié et plus compact que dans la version initiale. Une étape de plus peut être franchie en distribuant uniformément plusieurs ROM parmi les PE afin de limiter encore plus la distance à parcourir des signaux décodés. Par contre, le désavantage de cette approche est une plus grande utilisation des ressources du FPGA. Pour apporter un niveau de flexibilité supérieur lors de l'implantation, le rapport du nombre de ROM de décodage au nombre de PE a été paramétré, ce qui a permis d'effectuer différentes simulations afin de déterminer le meilleur rapport entre la distribution des signaux et l'utilisation des ressources.

Les résultats de complexité et de fréquence d'opération obtenus pour différentes configurations de décodeurs sont reproduits au Tableau 4.2. Le circuit utilisé pour effectuer ces comparaisons est un chemin de données composé de 8 PE de 16 bits, tels que présentés dans le chapitre précédent. Les quatre cas testés couvrent les principales possibilités comprises entre un décodeur unique et un décodeur par PE. Fait remarquable, le nombre de décodeur ne semble pas avoir d'influence sur la complexité et la fréquence

d'opération, ce qui peut s'expliquer par un partage des ressources effectué automatiquement par l'outil de synthèse et par l'insertion de tampons affectant la complexité. Cette optimisation laisse supposer qu'un seul décodeur pour toutes les configurations demeure une solution acceptable, étant donné le haut niveau d'interconnexions présentes pour le routage des ressources à l'intérieur du FPGA visé.

Tableau 4.2 Influence du nombre de décodeurs

Nombre de décodeurs	Fréquence maximale (MHz)	Complexité (Nombre de CLB)
1	42,2	4718
2	42,0	4748
4	41,7	4772
8	42,0	4748

4.1.5 Modifications des bus

Dans la version initiale du chemin de données, chaque module peut transmettre ses sorties sur différents bus, à l'aide des sélecteurs de bus décrits à la section 4.1.1. Tout dépendant des modules, ces écritures se font soit sur les bus de source, soit sur les bus de destination. En effet, les trois bus d'entrée de données, les mémoires, les registres et l'accumulateur communiquent leurs sorties par les bus de source, tandis que le décaleur barillet, le multiplicateur-sommateur et l'ALU inscrivent leurs données de sortie sur les bus de destination. Cet aiguillage des données vers les différents bus possibles est représenté à la Figure 4.5 par les matrices de sélection de source et de destination.

Cependant, dans la version originale du PE, la couverture de connexions n'était pas complète, ce qui entraînait qu'une sortie n'avait pas accès à tous les bus de source ou de destination. Certaines sorties pouvaient être reliées à plus d'un bus, tandis que d'autres sorties n'étaient connectées qu'à un seul bus en tout temps.

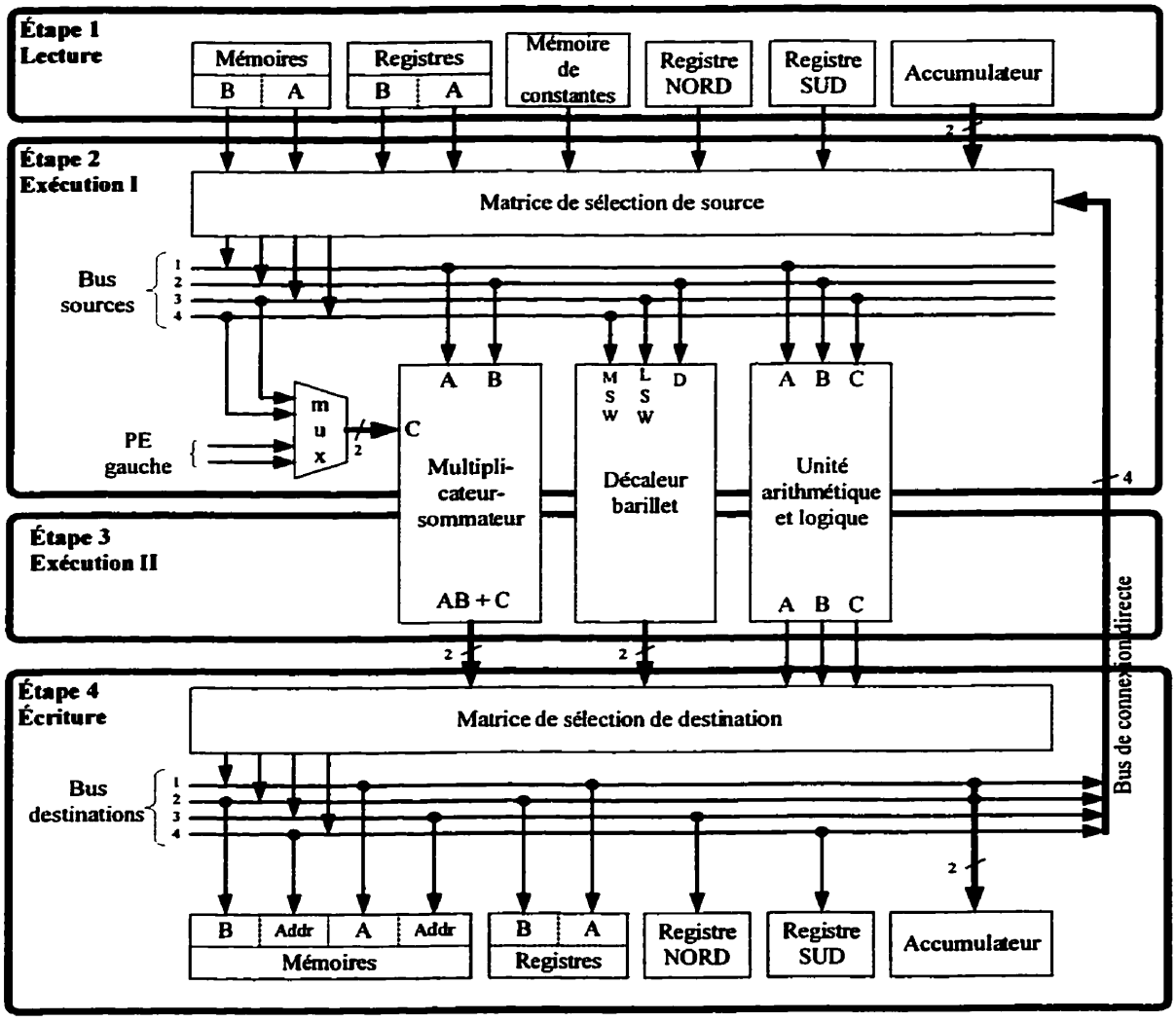

Figure 4.5 Architecture interne des PE modifiés et optimisés

Afin d'amener un mode d'adressage plus complet dans la nouvelle version du PE, le nombre de sélecteurs de bus a été augmenté pour chacune des sorties de telle sorte qu'elles puissent être connectées à tous les bus de source ou de destination, dépendant du module. Par exemple, les modifications apportées permettent maintenant à la première mémoire d'accéder à tous les bus de source alors qu'elle ne pouvait écrire que sur les trois premiers bus dans la version initiale. De façon similaire, la seconde sortie de l'ALU peut dorénavant transmettre des données sur tous les bus de destination tandis qu'elle n'était en mesure que d'inscrire des valeurs sur le deuxième et le quatrième bus dans l'ancien PE. Ce nouveau mode d'adressage est dit orthogonal car toutes les connexions entre les sorties de modules et les bus de source ou de destination sont permises.

La seconde modification effectuée sur la structure interne du PE concernant les bus est l'ajout d'une connexion directe des bus de destination vers les bus de source. Ce lien permet dorénavant de relier tous les bus de destination à n'importe lequel des bus de source. Chaque bus de destination est donc interprété comme une sortie de module pouvant s'appliquer à tous les bus de source. Malgré toute la souplesse apportée par ces nouvelles fonctionnalités, le contrôleur doit s'assurer qu'un seul signal ne parvient à chaque bus en tout temps, afin d'éviter les conflits d'écriture. En permettant plus d'itinéraires différents pour les transferts de données, ces améliorations de l'architecture des bus offre plus de flexibilité quant à l'implantation des algorithmes de traitement de signaux dans le processeur, une fois celui-ci réalisé.

4.1.6 Résultats des optimisations et modifications de l'architecture

Cette section vise à déterminer l'impact de l'ensemble des modifications et des optimisations précédentes sur l'implantation du circuit. Les deux paramètres importants examinés ici sont la complexité, mesurée en nombre de CLB, et la fréquence d'opération. Le Tableau 4.3 affiche les résultats obtenus pour les différentes versions du chemin de données et des PE. Les PE sont adaptés pour des mots de 16 bits pour les deux versions afin de comparer des PE ayant la même fonctionnalité. Pour la version optimisée, on observe une diminution de 30,7% des ressources utilisées pour une augmentation de 83,8% de la fréquence d'opération.

La fréquence d'opération des chemins de données est sensiblement la même que celle des PE étant donné la disposition parallèle de ceux-ci. En conséquence, les estimés de fréquence d'opération n'ont pas été générés. Les chemins de données comparés comportent 4 PE de 16 bits complets dans les deux versions et la nouvelle version affiche une baisse de 13,9% de la complexité par rapport à l'ancienne.

Tableau 4.3 Comparaison des versions du chemin de données et des PE

	Nombre de CLB	**Portion du FPGA utilisée (%)**	**Fréquence d'opération (MHz)**
Ancien PE	754	16.02	14.335
Nouveau PE	577	12.27	26.341
Ancien chemin de données	3002	63.82	-
Nouveau chemin de données	2636	56.03	-

4.2 Configuration du chemin de données

Comme son nom l'indique, cette section traite de la configuration de l'architecture du chemin de données. Cet objectif principal a été atteint grâce à différentes stratégies de programmation du code VHDL visant à rendre souple le modèle initial rigide. On peut scinder toutes les possibilités d'ajustement ajoutées en deux niveaux de configuration distincts : la paramétrisation et la modularité. La paramétrisation ne change pas l'architecture de haut niveau du chemin de données en ce sens qu'elle permet simplement d'adapter celui-ci à une application particulière. Par exemple, la paramétrisation permet entre autres le choix de la longueur des mots de donnée, du nombre de bus ou de la capacité des mémoires. Par ailleurs, le second niveau de configuration, la modularité, effectue des transformations de l'architecture de haut niveau du chemin de données. En effet, la modularité permet d'exclure, en totalité ou en partie, certains modules des PE. Une fois ces blocs retirés, la structure restante doit demeurer fonctionnelle, ce qui implique que tous les signaux connectés sur un module doivent être considérés en conséquence une fois celui-ci exclu de l'architecture. En terminant, il est important de spécifier que toutes les modifications s'appliquant à la structure du chemin de données sont réalisées lors de la synthèse. Ainsi, tout changement sur un circuit implanté doit être fait en re-synthétisant le code VHDL du processeur avec les nouveaux paramètres de configuration, pour ensuite reprogrammer le nouveau circuit ainsi créé dans le FPGA.

En plus de décrire les différentes options de configuration possibles du chemin de données, cette section présente une généralisation du protocole de développement adopté pour amener une méthode qualitative de migration d'un circuit ASIC vers un FPGA Virtex. Les différentes étapes à suivre pour rendre ce circuit configurable sont ensuite proposées. Une des phases de développement effectuée est d'ailleurs traitée de façon distincte et elle consiste à créer des types globaux pour la programmation du code VHDL.

4.2.1 Utilisation de types globaux

Étant donné la complexité du chemin de données et toutes les fonctionnalités supportées, plusieurs signaux de contrôle sont nécessaires pour la coordination de tous les modules. Cette multitude de signaux provenant du contrôleur est ensuite distribuée aux blocs fonctionnels concernés. L'utilité de ces signaux varie entre l'activation des modules, la sélection des modes de fonctionnement, le choix des bus sur lesquels les sorties seront envoyées, les adresses des mémoires et de registres ou encore la provenance de certaines entrées.

Dans la version initiale du chemin de données, tous ces signaux sont définis comme étant des bits (std_logic) ou des vecteurs de bits de différentes longueurs (std_logic_vector(x downto 0)). Cette structure de données est évidemment très simple à utiliser, mais a pour désavantage principal d'individualiser chaque signal. En effet, ceux-ci, même s'ils sont

tous connectés au même module, doivent être traités individuellement dans le code VHDL. De plus, les différents mode de fonctionnement ou tout autre paramètre de contrôle ne peuvent être exprimés de façon directe et explicite dans le code VHDL, car le concepteur doit utiliser la correspondance entre ceux-ci et un vecteur de 1 et de 0. Cette traduction à faire entre les valeurs possibles des signaux de contrôle et leurs représentations en vecteurs binaires rend la manipulation de ces signaux plus délicate et cause inévitablement des erreurs.

Pour remédier à ces inconvénients, nous avons eu recours à l'utilisation de types définis spécialement pour les différents signaux de contrôle. En créant un fichier de description des types inclus en tant que librairie de chaque module du processeur, ces types deviennent accessibles pour tout le code VHDL du processeur. Les définitions n'ont donc pas à être répliquées au début de chaque fichier. La structure de types privilégiée est le record qui permet un regroupement et une hiérarchie de tous les signaux communs à un module. Cette organisation simplifie énormément l'interface des blocs fonctionnels en diminuant le nombre de signaux de contrôle d'entrée à un seul pour la nouvelle version. Ce signal unique conserve toutefois la même fonctionnalité dû au fait qu'il est composé des mêmes signaux préalablement utilisés de façon individuelle. De plus, en ayant recours à des types explicites plutôt qu'à des vecteurs binaires, les modes de fonctionnement et tous les autres paramètres de contrôle sont exprimés directement en mots, sans avoir à trouver la correspondance avec une série de 1 et de 0. Cette méthode

de programmation est beaucoup plus intuitive et diminue sensiblement le nombre d'erreurs.

Bien que l'emploi de types hiérarchiques adaptés à chaque design est une stratégie de conception recommandable en tout temps, elle est d'autant plus importante pour rendre un modèle configurable. En simplifiant les interfaces entre les modules et le contrôleur, le nombre de signaux à paramétrer est réduit et l'exclusion de modules est aussi plus simple à gérer en ayant moins d'entrées et de sorties flottantes à ajuster. De plus, un signal peut être transformé ou paramétré de façon globale, simplement en modifiant la description du type correspondant dans le fichier de bibliothèque. Ce niveau de flexibilité additionnel permet d'implanter de nouvelles fonctionnalités plus rapidement et plus efficacement que dans la version originale où il aurait fallut modifier individuellement tous les modules utilisant ce signal.

4.2.2 Paramétrisation

Les modifications apportées au code VHDL du chemin de données utilisent principalement les clauses *if ... generate* et *for ... generate*. La première clause permet l'instanciation conditionnelle de structures selon la valeur d'expressions booléennes référant à certains paramètres. La seconde est utile pour générer une structure répétitive dont le nombre d'occurrences dépend d'une expression référant à certains paramètres. La

combinaison de ces deux clauses offre la flexibilité nécessaire pour la configuration de l'architecture avec une série de paramètres numériques.

Tous les paramètres de configuration sont inclus dans le fichier de types utilisé sous forme de bibliothèque dans chaque module. En procédant ainsi, chaque paramètre est visible dans tout le code VHDL du chemin de données. Lors de la configuration d'une architecture souhaitée, le concepteur n'a qu'à ajuster les valeurs de tous les paramètres à cet endroit unique et la synthèse de tous les modules s'effectue automatiquement en conséquence et ce, de manière totalement transparente. Voici la liste de tous les paramètres concernant le chemin de données et l'influence qu'ils ont sur la structure du circuit synthétisé.

- PE_Data_Width_c : Le paramètre le plus important et touchant tous les composants du chemin de données, il contrôle la longueur des mots de données traités dans chaque PE. La version initiale ne supportait que des données de 16 bits. La valeur minimale permise pour ce paramètre est de 6 bits mais aucune limite supérieure n'est fixée.

- Num_PE_c : Détermine le nombre de PE constituant le chemin de données. Initialement constant à 4 dans la version originale, la quantité de PE peut être ajustée pour emplir tout l'espace disponible du FPGA ciblé afin de permettre une capacité de traitement optimale.

- PE_Registers_Depth_c : Permet d'ajuster la capacité ou la quantité de mots pouvant être sauvegardés dans chacun des deux registres. La version originale du PE utilisait une profondeur de 32 mots de données pour chaque registre.

- PE_Memories_Depth_c : Tout comme le paramètre précédent, celui-ci s'applique cependant aux mémoires en y contrôlant la capacité permise en nombre de mots. Cette valeur était fixe à 256 mots pour la version originale.

- PE_Activity_Depth_c : Affecte la profondeur de la pile d'activité en modifiant ainsi le nombre boucles imbriquées supportées. La pile possédait une profondeur de 32 boucles dans la version originale du PE.

- Bus_Decoder_Ratio_c : Nouvelle fonctionnalité implantée, ce paramètre correspond au nombre de PE connectés sur une seule ROM de décodage. Le nombre de ROM synthétisées depend donc de cette valeur et du nombre de PE demandé. Voir la section 4.1.4 pour plus de détails.

La structure la plus complexe à rendre paramétrable est le multiplicateur composant le module multiplicateur-sommateur et c'est pourquoi le chapitre 2 lui a été dédié au complet. Sa difficulté vient du fait que son architecture est étroitement liée à la longueur de ses opérandes. Ainsi, pour chaque longueur distincte de mots, toute l'architecture doit

être régénérée différemment et les étages de pipeline ajustés en conséquence. Toute la synthèse du multiplicateur est réalisée à l'aide de relations mathématiques basées sur la largeur des mots de données (PE_Data_Width_c) et de paramètres exclusifs au multiplicateur dédiés à la latence et la configuration du pipeline.

4.2.3 Modularité

La modularité du chemin de données consiste en la possibilité d'inclure ou non certaines fonctionnalités dans le circuit synthétisé. Chaque PE est constitué de blocs fonctionnels dédiés à différentes fonctions. En étant principalement disposés de façon parallèle, la plupart de ces modules fonctionnent de façon relativement indépendante les uns par rapport aux autres. Ainsi, il devient facile de supprimer certaines fonctionnalités du modèle, simplement en enlevant les modules correspondant de l'architecture du circuit, sans affecter le fonctionnement des autres modules conservés.

Cette flexibilité est rendue possible grâce à l'utilisation de la clause *if ... generate* dans le code VHDL du PE. Celle-ci permet l'instanciation d'un module seulement lorsque la condition testée s'avère vraie. Le contrôle de l'inclusion ou de l'exclusion des blocs est alloué par une série de paramètres booléens, regroupés dans un fichier qui est accédé sous forme de librairie. Cette approche est la même que celle employée pour la paramétrisation. Pour configurer l'architecture des PE, le concepteur n'a qu'à attribuer faux aux paramètres correspondant aux modules à exclure et vrai aux paramètres liés aux

modules à inclure. Les dépendances entre certains modules sont gérées automatiquement pour éviter d'exclure par erreur une fonctionnalité qui serait nécessaire à un autre module utilisé dans l'architecture. Voici les différents paramètres booléens requis pour la modularité des PE et leurs effets sur la structure.

- Barrel_Shifter_Included : Contrôle la présence ou non du décaleur barillet dans l'architecture du PE.

- Constants_and_Forwarding_Included : Contrôle simultanément la présence ou non dans l'architecture des PE, de connexions directes, en provenance des bus de destination et du bus d'entrée de constantes, vers les bus de source.

- Constants_Included : Contrôle exclusivement la présence ou non dans l'architecture des PE, des connexions du bus de constantes vers les bus de source.

- Forwarding_Bus1_Included, Forwarding_Bus2_Included, Forwarding_Bus3_Included et Forwarding_Bus4_Included : Contrôlent individuellement la présence ou non dans l'architecture des PE, des connexions de chaque bus de destination vers les bus de source.

- Accumulator_Included : Contrôle la présence ou non de l'accumulateur dans l'architecture des PE.

- MADD_Unit_Included : Contrôle la présence ou non du multiplicateur-sommateur dans l'architecture des PE.

- Arithmetic_Logic_Unit_Included : Contrôle la présence ou non de l'ALU en entier dans l'architecture des PE.

- Arith_Functions_Included : Contrôle la présence ou non des structures internes de l'ALU supportant les fonctions arithmétiques.

- Logic_Functions_Included : Contrôle la présence ou non des structures internes de l'ALU supportant les fonctions logiques.

- Three_Point_Swap_Included : Contrôle la présence ou non des structures internes de l'ALU supportant les permutations des trois entrées. L'inclusion de cette structure de permutation nécessite la présence des fonctions arithmétiques qui sont alors automatiquement intégrées à l'ALU, indépendamment de la valeur de Arith_Functions_Included.

- Memory_Blocks_Included : Contrôle la présence ou non des deux mémoires dans l'architecture des PE.

- MemB_Included : Contrôle la présence ou non de la seconde mémoire dans l'architecture des PE.

- Register_Files_Block_Included : Contrôle la présence ou non des deux registres dans l'architecture des PE.

- RfB_Included : Contrôle la présence ou non du second registre dans l'architecture des PE.

Il est important de s'assurer que lors de l'exclusion d'un module, tous les signaux d'entrée et de sortie qui y étaient attachés sont gérés convenablement. En d'autres termes, le fait de retirer un module ne doit pas affecter les blocs voisins. L'outil de synthèse Synplify facilite cette tâche en supprimant automatiquement tout signal d'entrée d'un module exclu et en forçant tous ses signaux de sortie à des valeurs nulles dans le cas de vecteurs ou de bits, à faux pour les signaux de type booléens et à la première valeur d'une liste d'énumération en ce qui concerne les signaux de sortie de ce type. Pour les cas où ces valeurs attribuées par défaut ne conviendraient pas, le concepteur doit assigner les valeurs voulues à ces signaux à l'intérieur d'un clause *if ... generate* dont la condition vérifie l'absence de ce module dans l'architecture.

4.2.4 Résultats de la paramétrisation et de la modularité

Afin de déterminer l'impact de la paramétrisation et de la modularité décrites dans les sections 4.2.2 et 4.2.3, différents essais de configuration ont été réalisés. Pour ce faire, le paramètre étudié doit être modifié pour ensuite synthétiser le circuit équivalent à la configuration demandée. L'étape suivante consiste à effectuer le placement et le routage de ce circuit dans le FPGA Virtex XCV800 visé pour l'implantation. Les résultats observés comprennent le nombre de CLB utilisés ainsi que le pourcentage de ressources disponibles dédiées au circuit. Tous ces résultats proviennent de l'outil de placement-routage Alliance de Xilinx.

Le Tableau 4.4 montre différentes configurations de PE obtenues en retranchant chaque module à l'aide des clauses d'inclusion-exclusion décrites dans la section 4.2.3. Le nombre de combinaisons possibles nous limite à l'étude individuelle de l'influence de chaque module. Cependant, l'absence de plusieurs modules est tout simplement l'addition de l'économie des ressources réalisée pour chacun des modules. En moyenne, chaque module exclu de la structure lors de la synthèse du circuit permet une économie de 9,3% des ressources du FPGA et peut aller jusqu'à 38,4% pour l'exclusion des mémoires (*memory blocks*). Un fait étrange apparaît pour l'exclusion des bus de connexions directes et du bus de constantes. En effet, le nombre de CLB est plus élevé lorsque ces composantes sont exclues de l'architecture du PE, ce qui s'explique par une optimisation par l'outil de synthèse Synplify, qui n'est possible que lorsque les bus de constantes et de connexions directes sont présents.

Tableau 4.4 Influence de la modularité sur la complexité

Module exclu	Nombre de CLB	Portion du FPGA utilisée (%)
aucun (PE complet)	577	12.27
barrel shifter	500	10.62
constants and forwarding	577	12.27
constants	580	12.32
forwarding bus 1	584	12.40
forwarding bus 1, 2, 3 et 4	586	12.45
accumulator	547	11.63
madd unit	463	9.84
arithmetic logic unit	475	10.10
arith functions	577	12.27
logic functions	567	12.05
three point swap	531	11.28
three point swap et arith functions	519	11.02
memory blocks	417	8.86
memb	502	10.67
register files block	539	11.45
rfb	563	11.96

L'étude suivante consiste à analyser l'influence de la longueur des mots traités par le PE sur la complexité, tel que présenté dans le Tableau 4.5. Pour ce faire, l'équation (4.1) a été développée en utilisant la méthode des moindres carrés. Cette dernière permet de trouver les coefficients d'un polynôme représentant le comportement de la complexité selon la longueur des mots.

Tableau 4.5 Influence de la longueur des mots du PE sur la complexité

Longueur des mots (bits)	Nombre de CLB	Portion du FPGA utilisée (%)
8	303	6.43
10	394	8.37
16	577	12.27
20	779	16.56
24	900	19.12
32	1316	27.97
64	3250	69.08

$$\text{Nombre de CLB} = 0.3488(\text{longueur})^2 + 27.4075(\text{longueur}) + 68.1730 \qquad (4.1)$$

Le Tableau 4.6 détermine l'impact du nombre de PE sur la complexité d'un chemin de données. Les PE sont tous complets et conçus pour des mots de 16 bits. Seuls les cas les plus représentatifs ont étés relevés, étant donné le temps requis pour la synthèse des chemins de données à plusieurs PE, ainsi que la complexité des circuits générés, dépassant la capacité du Virtex XCV800 visé. Toutefois les exemples ci-dessous nous permettent de trouver une relation (équation (4.2))entre la complexité et le nombre de PE, en ayant recours à la méthode des moindres carrés, décrite précédemment.

Tableau 4.6 Influence du nombre de PE du chemin de données sur la complexité

Nombre de PE (16 bits)	Nombre de CLB	Portion du FPGA utilisée (%)
1	577	12.27
2	1224	26.02
4	2636	56.03

$$\text{Nombre de CLB} = 19.6667(\text{nombre PE})^2 + 588(\text{nombre PE}) - 30.6667 \qquad (4.2)$$

4.2.5 Méthodologie de migration et d'adaptation pour la configuration

Toutes les étapes discutées préalablement dans ce chapitre peuvent être généralisées en une méthodologie de développement en vue de porter une description VHDL d'un circuit initialement conçu pour un ASIC vers un FPGA de la famille Virtex de Xilinx. Ces premières étapes permettent donc d'exploiter efficacement les ressources disponibles dans un FPGA Virtex :

- Supprimer toute implantation directe d'amplificateur sur un signal dans le but de permettre une sortance plus élevé. En effet, l'architecture du circuit programmable du FPGA Virtex est conçue pour faire face à ce genre de situation automatiquement, grâce aux outils de placement et de routage qui adaptent la structure en conséquence.

- Adapter les sélecteurs de bus de telle sorte qu'ils exploitent les tampons à trois états présents dans la structure des FPGA Virtex. Supprimer ensuite tous les multiplexeurs et les portes devenues inutiles.

- Supprimer tout circuit utilisé à l'implémentation d'une interface de test JTAG étant donné sa présence sur tous les FPGA Virtex, selon la norme IEEE 1149.1.

- Considérer l'utilisation des BlockRAM configurables afin de libérer les CLB pouvant être utilisés pour l'implémentation du circuit. De préférence, toute structure de mémoire devrait être portée vers les BlockRAM, étant donnée leur nombre et leur facilité d'utilisation.

- Transformer toute addition effectuées à l'aide de cellules d'addition à un bit disposées en cascade afin d'utiliser plutôt un simple signe d'addition « + » dans le code VHDL synthétisable. De plus, toutes les structures de sommation rapide telles que *carry-save, carry-look-ahead ou carry-ripple* doivent aussi être supprimées pour exploiter les chaînes de propagation de retenues rapides incluses dans chaque CLB. Celles-ci permettent une implantation beaucoup plus compacte et rapide de l'addition.

- Bien qu'elles n'aient pas été utilisées pour le développement du processeur concerné dans ce mémoire, plusieurs autres structures sont prévues afin de faciliter l'implantation de circuits complexes. On note la présence d'interfaces PCI, de 4

unités DLL (*Delay-Locked Loops*) pour la gestion de l'horloge, d'interfaces d'entrées et sorties configurables selon différents protocoles de communication, de circuits protecteurs de décharges électrostatiques sur chaque connecteur, d'une sonde de la température du silicium et de plusieurs autres caractéristiques intéressantes pour l'implantation de circuits.

Une seconde phase de développement peut succéder à la première afin de paramétrer et de rendre modulaire l'architecture en question. Voici les étapes suggérées pour atteindre un tel objectif.

Fractionner tous les blocs fonctionnels du circuit à rendre configurable en fichiers distincts qui sont beaucoup plus faciles à modifier et à valider qu'un seul fichier comprenant tous les modules employés dans le circuit.

Utiliser un fichier de type librairie pour la description de tous les types employés dans le code VHDL du circuit en développement. Autant que possible, regrouper les différents signaux de contrôle de chaque module en un seul signal. Pour ce faire, définir un seul type par module qui est alors composé d'autres types de façon hiérarchique, c'est-à-dire une structure de record. Aussi, utiliser des types explicites pour les signaux, plutôt que des vecteurs binaires sans signification directe. L'interface des différents blocs fonctionnels devient alors plus simple à gérer.

Paramétrer l'architecture du circuit en ayant recours aux clauses *if ... generate* et *for ... generate* afin d'automatiser la synthèse selon des conditions impliquant différents paramètres définis. Regrouper ces paramètres sous forme de constantes dans un seul fichier utilisé comme bibliothèque. Il est préférable de ne pas utiliser la structure de programmation *generic* car elle n'est visible qu'à l'intérieur du fichier où elle est définie.

Rendre modulaire le circuit en utilisant principalement la clause *if ... generate* qui permet d'instancier ou non un module selon le résultat de la condition. Dans ce cas, la condition peut simplement être un paramètre booléen qui doit être vrai pour l'inclusion et faux pour l'exclusion du module en question. Chaque module pouvant être exclu de l'architecture devra donc comporter un paramètre associé. Tous les paramètres seront ensuite regroupés en un seul fichier (pouvant être le même qu'à l'étape précédente), utilisé sous forme de bibliothèque dans les modules de haut niveau où sont effectuées les instanciations des modules ainsi contrôlés. Il faut aussi s'assurer que tous les signaux connectés à un module sont bien gérés lors de l'exclusion de celui-ci. Aussi, tenir compte des dépendances possibles entre certains blocs en ajoutant des conditions supplémentaires au besoin.

4.3 Validation du chemin de données

Comme dans tous les domaines de conception, des tests doivent être effectués pour s'assurer que le nouveau design correspond bien aux spécifications attendues. Cette étape

indispensable permet de détecter les problèmes potentiels et de les corriger avant la production finale. La version initiale du processeur comportait déjà un banc de test permettant de simuler les programmes de plusieurs applications afin de comparer ensuite les résultats générés à ceux attendus. Les programmes employés à cette fin exploitaient différentes fonctionnalités de manière à utiliser tous les blocs fonctionnels afin d'y détecter toute anomalie.

La première approche fut donc de récupérer ce banc de tests et de le réutiliser en remplaçant l'ancien chemin de données par le nouveau. Certains obstacles sont alors apparus. En effet, de nouveaux types hiérarchiques ayant été utilisé pour la nouvelle architecture, la communication entre l'ancien contrôleur et le nouveau chemin de données devenait impossible. Pour remédier à ceci, nous avons développé une interface de types complètement transparente au reste de l'architecture, c'est-à-dire n'insérant aucun délai supplémentaire et ne modifiant pas la signification des signaux. Cette interface, s'insérant entre le contrôleur et le chemin de données, convertit et regroupe les anciens signaux de contrôle en provenance du contrôleur, exprimés sous forme de vecteurs binaires, en signaux hiérarchiques définis selon les nouveaux types en usage dans le chemin de données. Le second problème venait du fait que la ROM de décodage est maintenant située dans le chemin de données plutôt que dans le contrôleur comme dans la version initiale. La solution adoptée fut d'inclure dans l'interface de types une ROM d'encodage ayant exactement l'effet inverse de la ROM d'encodage employée. Sa présence permet donc de coder les signaux de contrôle d'écriture des bus pour ainsi simuler la présence

d'un contrôleur n'effectuant pas la conversion et qui serait compatible avec le nouveau chemin de données. Par ailleurs, le nouveau contrôleur n'était pas encore disponible pour la simulation avec le modèle complet du processeur.

Ces modifications effectuées, l'utilité d'un tel banc de test s'est montrée très limitée et inadéquate au déverminage du chemin de données testé. Aucun signal interne n'étant visible, tout le fonctionnement interne devient complètement transparent pour le concepteur qui n'a alors recours qu'aux données de sortie insuffisantes pour valider le modèle.

La seconde solution envisagée fut adoptée pour la validation complète de tout le chemin de données. Un second banc de test a été développé spécialement pour l'architecture examinée. Pour simuler l'action du contrôleur, tous les signaux de contrôle sont gérés à l'aide d'un fichier lu au début de chaque simulation. Celui-ci comporte tous les modes de fonctionnement de chaque module ainsi que tous les autres signaux utilisés pendant l'exécution d'une application. La stratégie de validation employée fut de simuler les différents modes de communication entre les PE par les canaux nord et sud. Une fois complétée, cette phase de validation permit d'entrer à l'intérieur des PE pour simuler chaque module de façon indépendante. Pour ce faire, une simulation exécutait tous les modes de fonctionnement possibles avec des données critiques pour ensuite vérifier la validité des sorties. Ces étapes ont été réalisées pour tous les modules composant un PE pour les différents modes de fonctionnement les caractérisant et avec des données

susceptibles de créer des sorties erronées dans le cas d'un fonctionnement erroné de ces modules. En plus de la fonctionnalité de tous les modules, le synchronisme des données avec les différents niveaux de latence internes du PE a été validé.

La validation a permis de corriger quelques erreurs présentes dans le fichier de types ayant des répercussions sur la sélection des modes de fonctionnement de certains modules du PE. De plus, le décaleur barillet a dû être reprogrammé entièrement dû à plusieurs erreurs détectées. Finalement, l'écriture des données du bus de constantes et des mémoires ne respectant pas le niveau de pipeline attendu, des registres ont dû être insérés afin d'adapter la latence au reste de l'architecture du PE.

CONCLUSION

Le but du travail exposé dans ce mémoire est d'obtenir un chemin de données décrit sous forme de VHDL synthétisable. Ce chemin de données doit être paramétrable sous plusieurs aspects et avoir la possibilité d'inclure ou d'exclure certains modules du circuit final. La technologie visée est un FPGA reprogrammable, le modèle XCV800 de la famille Virtex de Xilinx. Le point de départ de ce travail était le code VHDL du processeur DSP PULSE de type SIMD, initialement développé pour une implantation dans un ASIC. Dans ce mémoire, nous avons donc exploré les différentes étapes nécessaires à la transformation du chemin de données à architecture fixe de PULSE, en un chemin de données dont la structure est configurable sous plusieurs aspects. De plus, plusieurs modifications ont dû être apportées au modèle original afin de l'adapter et l'optimiser pour le FPGA ciblé.

Le premier chapitre constitue une revue de littérature ayant pour objectif de présenter brièvement les différents travaux réalisés dans le domaine des processeurs configurables. Il existe présentement sur le marché deux principales compagnies, ARC et Tensilica, proposant des processeurs dont le niveau de configuration permet au concepteur une flexibilité de conception intéressante. En effet, chacun de ces deux produits est livré avec une suite d'outils de développement complète, réalisant toutes les étapes, allant du choix des spécifications, jusqu'aux simulations sur les systèmes de prototypes émulant les circuits finaux, en passant par la génération des compilateurs et du développement de logiciels embarqués. La seconde partie suggère différentes méthodes de développement

de circuits paramétrables qui sont proposées dans la littérature, ainsi que des exemples de processeurs et de chemins de données configurables. Le chapitre se termine par une brêve présentation des architectures reconfigurables, que ce soit de façon statique (la configuration est effectuée avant l'exécution de l'application) ou dynamique (la configuration de l'architecture est effectuée simultanément avec l'exécution d'une application).

Le deuxième chapitre expose un algorithme réalisé en VHDL, décrivant l'architecture d'un multiplicateur signé, pipeliné et configurable. Un simple paramètre permet de modifier le circuit synthétisé de telle sorte qu'il puisse accepter des opérandes de différentes longueur supérieures à 5 bits. De plus, le nombre d'étages de pipeline ainsi que leur disposition est configurable grâce à d'autres paramètres. Cette flexibilité en fait une architecture compatible avec n'importe quel design utilisant une représentation des nombres en complément à deux, peu importe la latence de celui-ci. Les résultats du placement et du routage pour diverses configurations de cette structure affichent des performances supérieures aux multiplicateurs fournis par Xilinx sous forme de bibliothèques. Cependant, la complexité des circuits générés par l'algorithme proposé est plus importante que celle des circuits de Xilinx, dû à une plus grande quantité de ressources requises lors de l'implantation.

Le troisième chapitre traite de l'architecture initiale du processeur PULSE, en insistant toutefois sur le chemin de données qui constitue le cœur de cet ouvrage. Une vue globale

de la structure de haut niveau du processeur est montrée en premier lieu, pour ensuite expliquer le fonctionnement du chemin de données et des PE qui le composent. Tous les modules internes sont décrits individuellement, avec leurs rôles et les différents modes de fonctionnement qui les caractérisent. Les bus de source et de destination sont aussi exposés car ce sont eux qui assurent toutes les communications de données entre les modules de traitement à l'intérieur des PE. La seconde partie du chapitre présente différents aspects de l'architecture des FPGA Virtex de Xilinx qui seront utilisés comme plate-forme finale du circuit. Différents avantages de ce choix par rapport aux ASIC sont pointés. La famille Virtex possède plusieurs caractéristiques intéressantes pour l'utilisation que nous souhaitons en faire. Plusieurs structures sont présentes sur ces FPGA afin de faciliter l'implantation circuits, particulièrement en ce qui concerne les mémoires, les fonctions arithmétiques et les bus partagés. La dernière section présente les outils logiciels utilisés pour chacune des phases du développement du chemin de données.

Le quatrième et dernier chapitre aborde chacune des étapes de développement en détails. Il est subdivisé en trois grandes sections. La première partie traite des modifications effectuées sur différents modules afin d'obtenir un modèle VHDL adapté et optimisé pour l'architecture Virtex. Le circuit doit tirer parti de toutes les structures intégrées dans ces FPGA, afin d'être plus compact et plus rapide. D'autres changements ont été faits sur quelques modules pour les améliorer et pour ajouter de la flexibilité au niveau des communications internes des PE. La section suivante permet de bien comprendre les

stratégies mises à profit pour rendre le chemin de données configurable. La nuance entre la paramétrisation et la modularité y est expliquée. Le premier niveau de configuration, la paramétrisation, ne change pas l'architecture de haut niveau du chemin de données mais adapte celle-ci à une application donnée en changeant la longueur des mots traités, la capacité des mémoires et registres, ou encore le nombre de PE par exemple. Par ailleurs, la modularité permet d'enlever certains modules de l'architecture lors de la synthèse, dans le but d'obtenir un circuit plus compact. Toutes les options disponibles, autant pour la paramétrisation que pour la modularité, sont décrites. Le chapitre poursuit en généralisant le travail effectué de manière à présenter une méthode de migration vers un FPGA Xilinx recourant à l'ajout d'options de configuration, qui soit applicable à tout design. Finalement, la stratégie de simulation mise en œuvre pour la validation du chemin de données est démontrée.

Le quatrième chapitre termine en présentant les résultats obtenus par la synthèse, le placement et le routage de différents circuits possédant diverses configuration. Ces résultats ont pour mandat de montrer les différences de complexité entre des circuits, selon les options de configuration sélectionnées. La première partie traite de la modularité en comparant l'impact de l'exclusion de chaque module individuellement. Cependant, la complexité peut être diminuée davantage lorsque plus d'un module est retranché du modèle. La seconde partie examine l'impact de la longueur des mots de données sur la complexité du PE. Une équation est présentée comme généralisation pour estimer la complexité d'un PE pour une longueur de mots donnée. Finalement, la

troisième partie des résultats considère l'influence du nombre de PE dans un chemin de données, afin d'obtenir une seconde équation permettant d'estimer la complexité d'un circuit à multiples PE.

Le travail qui reste à compléter sur le processeur configurable présenté consiste à finaliser l'analyseur de microcode qui détecte les modules requis par une application en examinant son programme (Hébert, 2000). Cet outil devra ensuite être en mesure de créer un fichier de paramètres qui sera utilisé lors de la synthèse du code VHDL, afin de générer le circuit adapté à l'application visée. Une caractéristique intéressante serait d'ajuster automatiquement le nombre de PE du chemin de données configuré, de manière à utiliser le maximum de ressources disponibles du FPGA et ainsi obtenir une capacité de traitement optimale. La capacité de ce processeur à comporter un chemin de données dont le nombre de PE est variable est sans doute l'aspect de paramétrisation le plus intéressant. En effet, l'ajout de PE permet un gain de performance notable, et aucun des processeurs configurables examinés ne possède cette caractéristique.

En conclusion, ce mémoire démontre les multiples avantages liés à l'utilisation d'un processeur configurable, par rapport aux processeurs conventionnels. En effet, les modèles configurables permettent un niveau de flexibilité, allouant une plus grande marge de manœuvre aux concepteurs qui peuvent alors optimiser beaucoup plus efficacement les circuits. Le domaine des processeurs embarqués est particulièrement adapté aux processeurs configurables, du fait qu'ils ne nécessitent pas une grande

versatilité. Ils peuvent donc être spécialisés et optimisés pour une tâche particulière. L'étape suivante à franchir est le développement de processeurs reconfigurables qui exploiteront plus efficacement les ressources disponibles à l'intérieur des FPGA et qui implanteront directement en matériel les algorithmes à effectuer, de manière à offrir des performances supérieures de plusieurs ordres de grandeur à ce que l'on connaît aujourd'hui.

RÉFÉRENCES

[1] Active-HDL Standard Edition Datasheet. Aldec, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Aldec, Inc.* [En ligne]. http://www.aldec.com/pages/StandardEdition.htm

[2] Alliance Series v2.1i Software. Xilinx, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Xilinx, Inc.* [En ligne]. http://www.xilinx.com/products/alliance.htm

[3] ARC Cores, Ltd. (Page consultée le 26 mars 2000). *Site de la compagnie ARC Cores, Ltd.* [En ligne]. http://www.arccores.com/

[4] ASHENDEN, P.J. (1996). The Designer's Guide to VHDL, Morgan Kaufmann Publishers, Inc., San Francisco, 688 p.

[5] BAUGH, C.R. et WOOLEY, B.A. (décembre 1973). A Two's Complement Parallel Array Multiplication Algorithm. IEEE Transactions on Computers, vol. C-22, no. 12, 1045-1047.

[6] BE'ERY, Y., BERGER, S. et OVADIA, B.-S. (octobre 1993). An application-specific DSP for portable applications. Proceedings of IEEE Workshop on VLSI Signal Processing, 48-56.

[7] BEREKOVIC, M., HEISTERMANN, D. et PIRSCH, P. (octobre 1998). A core generator for fully synthesizable and highly parameterizable RISC-cores for system-on-chip designs. IEEE Workshop on Signal Processing Systems, SIPS 98 : Design and Implementation, 561-568.

[8] BHATIA, D. (janvier 1997). Reconfigurable computing. Proceedings. Tenth International Conference on VLSI Design, 356-359.

[9] CAVANAGH, J.J.F. (1984). Digital Computer Arithmetic: Design and Implementation, McGraw-Hill, New-York, 468 p.

[10] Core Solutions – Base-Level Products, Math Functions. Xilinx, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Xilinx, Inc.* [En ligne]. http://www.xilinx.com/products/logicore/tbl_base_lvl.htm#math

[11] CRONQUIST, D.C., FISHER, C., FIGUEROA, M., FRANKLIN, P. et EBELING, C. (mars 1999). Architecture design of reconfigurable pipelined datapaths. Proceedings 20th Anniversary Conference on Advanced Research in VLSI, 23-40.

[12] DYCK, A., EVENSON, S., FU, H. et HOBSON, R. (juin 1999). User selectable feature support for an embedded processor. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI, vol. 1, 9-12.

[13] EBELING, C., CRONQUIST, D.C. et FRANKLIN, P. (juillet 1997). Configurable computing: the catalyst for high-performance architectures. Proceedings. IEEE, International Conference on Applications-Specific Systems, Architectures and Processors, 364-372.

[14] FAWCETT, B.K. (novembre 1995). FPGAs in reconfigurable computing applications. Proceedings of WESCON95, 261-266.

[15] GAJJALAPURNA, K.M. et BHATIA, D. (juin 1998). Emulating large designs on small reconfigurable hardware. Proceedings. Ninth International Workshop on Rapid System Prototyping, 58-63.

[16] GRAYVER, E. et DANESHRAD, B. (juin 1998). Reconfigurable signal processing ASIC architecture for high speed data communications. Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, vol. 4, 389-392.

[17] HDL Simulation Products. Model Technology, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Model Technology, Inc.* [En ligne]. http://www.model.com/products/index.html

[18] HÉBERT, O. (octobre 2000). Une méthode de dérivation de modèles de processeurs embarqués dédiés à une application, et un modèle de processeur de traitement de signal conçu pour l'implanter. Mémoire de maîtrise, École Polytechnique de Montréal, Montréal.

[19] HOGL, H., KUGEL, A., LUDVIG, J., MANNER, R., NOFFZ, K.H. et ZOZ, R. (avril 1995). Enable++: a second generation FPGA processor. Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 45-53.

[20] How Much Configurability Do You Need? Issues Forum, Silicon Strategies. (août 1999). (Page consultée le 26 mars 2000). *Site du magazine Silicon Strategies.* [En ligne]. http://www.s2mag.com/Editorial/1999/issuesforum9908.html

[21] JIN-HYUK Y., BYOUNG-WOON K., SUNG-WON S., SANG-JUN N., CHANG-HO R., JANG-HO C. et CHONG-MIN K. (février 1998). MetaCore : a configurable & instruction-level extensible DSP core. Proceedings of the ASP-DAC '98 Asian and South Pacific Design Automation Conference 1998, 325-326.

[22] KUULUSA, M., NURMI, J., TAKALA, J., OJALA, P. et HERRANEN, H. (octobre-décembre 1997). A flexible DSP core for embedded systems. IEEE Design & Test of Computers, vol. 14, 60-68.

[23] LEMOINE, E. et MERCERON, D. (avril 1995). Run time reconfiguration of FPGA for scanning genomic databases. Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 90-98.

[24] LEVY, M. (7 janvier 1999). Customized Processors : Have it Your Way. EDN, 97-104.

[25] Market Background. Lexra, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Lexra, Inc.* [En ligne]. http://www.lexra.com/lx_marketbg2.html

[26] MARRIOTT, P., KRALJIC, I.C. et SAVARIA, Y. (octobre 1998). Parallel ultra large scale engine SIMD architecture for real-time digital signal processing applications. Proceedings International Conference on Computer Design : VLSI in Computers and Processors, 482-487.

[27] MATSUMOTO C. (4 octobre 1999). Reconfigurable computing to take first space test. Electronic Engineering Times, 14.

[28] MCCANNY, J.V., HU, Y., DING, T.J., TRAINOR, D. et RIDGE, D. (novembre 1996). Rapid design of DSP ASIC cores using hierarchical VHDL libraries. Thirtieth Asilomar Conference on Signals, Systems and Computers, vol. 2, 1344-1348.

[29] MCCLUSKEY, J. Practical Applications of Recursive VHDL Components in FPGA Synthesis. Lucent Technologies, Inc. (Page consultée le 26 mars 2000). *Site de Lucent Technologies Canada, Inc.* [En ligne]. http://www.lucent.ca/fpga/papers.htm

[30] NURMI, J. et TAKALA, J. (novembre 1997). A new generation of parameterized and extensible DSP cores. IEEE Workshop on Signal Processing Systems, SiPS 97 : Design and Implementation formerly VLSI Signal Processing, 320-329.

[31] Product Literature. Synplicity, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Synplicity, Inc.* [En ligne]. http://www.synplicity.com/literature/index.html

[32] RAMANATHAN, S., VISVANATHAN et V., NANDY, S.K. (janvier 1999). Synthesis of configurable architectures for DSP algorithms. Proceedings Twelfth International Conference on VLSI Design, 350-357.

[33] SMITH, S.G., MORGAN, R.W. et PAYNE, J.G. (octobre 1989). Generic ASIC architecture for digital signal processing. Proceedings. 1989 IEEE International Conference on Computer Design : VLSI in Computers and Processors, 82-85.

[34] Tensilica, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Tensilica, Inc.* [En ligne]. http://www.tensilica.com/home.html

[35] Virtex product specifications. Xilinx, Inc. (Page consultée le 26 mars 2000). *Site de la compagnie Xilinx, Inc.* [En ligne]. http://www.xilinx.com/partinfo/ds003.pdf

[36] WESTFELD W. (20 septembre 1999). Hardware upgraded remotely via nets. Electronic Engineering Times, 98.

[37] WIRTHLIN, M.J., HUTCHINGS, B.L. et GILSON, K.L. (avril 1994). The Nano Processor: a low resource reconfigurable processor. Proceedings IEEE Workshop on FPGAs for Custom Computing Machines, 23-30.