UNIVERSITÉ DE MONTRÉAL

CONCEPTION ET MISE EN ŒUVRE D'UN SYSTÈME DE RECONFIGURATION DYNAMIQUE

CYNTHIA COUSINEAU

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)

JANVIER 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION ET MISE EN OEUVRE D'UN SYSTÈME DE RECONFIGURATION DYNAMIQUE

présenté par : COUSINEAU Cynthia
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. HOULE Jean-Louis, Ph.D., membre

REMERCIEMENTS

La réalisation de ce projet de maîtrise n'aurait pu être possible sans l'aide de Monsieur Yvon Savaria, mon directeur de recherche, qui m'a proposé ce sujet et qui m'a soutenu et aidé tout au long de son développement ainsi que Monsieur Mohamad Sawan, mon co-directeur de recherche. Également, je tiens à remercier Monsieur Pierre Popovic et toute l'équipe de MiroTech Microsystems Inc. qui m'ont fourni ressources et supports pour le développement de ce projet ainsi qu'un milieu de travail propice à la conception. En dernier lieu, je tiens à remercier Madame Virginie Cousineau qui a critiqué de manière constructive ce mémoire ainsi que mon conjoint et mes parents qui m'ont supporté moralement durant ce travail. Merci.

RÉSUMÉ

Ce mémoire traite de la reconfiguration dynamique des FPGA. La reconfiguration dynamique, ou RTR pour *Run Time Reconfiguration*, est une technique qui permet de conserver une partie d'un système en traitement pendant que l'autre se fait reconfigurer. Cette nouvelle approche de reconfiguration se divise principalement en deux champs, soit la reconfiguration dynamique des FPGA reprogrammables partiellement et la reconfiguration dynamique utilisant des FPGA conventionnels. Ces deux branches de la reconfiguration dynamique sont passablement différentes et apportent des avantages et inconvénients distincts. Le travail de recherche qui a été effectué pour ce mémoire repose sur la reconfiguration dynamique utilisant les FPGA conventionnels. Toutefois, nous avons tout de même démontré les points que nous tentons d'améliorer par rapport à l'autre technique de reconfiguration dynamique, celle utilisant les FPGA reconfigurables partiellement.

Le système proposé a été implanté sur une carte, commercialement disponible, du nom de X-C436 et produite par MiroTech Microsystems Inc. de Ville St-Laurent. La compagnie nous a approché pour tenter l'implantation d'un système de reconfiguration dynamique sur cette carte moyennant l'utilisation d'un lien de communication secondaire, le lien sériel JTAG, régit par le protocole IEEE 1149.1 (*Boundary Scan*). Dans ce projet, on utilise la technique de reconfiguration dynamique avec des FPGA conventionnels, puisque la carte est munie de ce type de FPGA, des XC4036EX de Xilinx. La modification de la carte est la partie principale de ce projet. Elle consistait en un remaniement du contrôleur de la carte, le CSU, sigle de *Configuration and Shutdown Unit*. Ce contrôleur est, quant à lui, un FPGA de la famille XC3000 de Xilinx.

Le concept de reconfiguration dynamique de la carte repose sur le fait que celle-ci possède deux unités de traitement. Ainsi, en utilisant toujours une de ces unités pour effectuer un traitement, nous pouvons reconfigurer la deuxième pour que celle-ci soit en mesure de poursuivre le traitement débuté par la première. Le type de système qui peut bénéficier de la reconfiguration dynamique est un système qu'on peut décortiquer en plusieurs séries de traitements. Ces traitements unitaires sont par la

suite implantés, un à un, dans les unités de traitement. On réalise ainsi une grande économie matérielle grâce à la reconfiguration dynamique.

Une fois que l'implantation matérielle fut réalisée, nous avons procédé à une démonstration qui nécessitait également l'implantation d'une partie logicielle. Étant donné que la carte X-C436 est conçue pour être utilisée avec des DSP TMS320C4x de Texas Instruments, la démonstration réalisée utilisait deux de ces DSP pour effectuer un traitement d'image continuel. En effet, la démonstration consistait en l'application, sur une image vidéo noir et blanc, d'une série de filtres Sobel, dont la progression créait l'effet d'une rotation dans l'angle de projection d'une détection de contour.

L'implantation nous a permis de mesurer exactement le temps de reconfiguration nécessaire pour chaque filtre. C'est à l'aide du lien JTAG que nous transmettions le fichier de configuration de l'unité de traitement à configurer. La vitesse de ce lien sériel était de 8.25 MHz en théorie, ce qui nous aurait permis d'effectuer une reconfiguration complète, incluant le temps de permutation entre les deux unités traitantes, en deçà de 100 ms. Cependant, les mesures faites nous ont malheureusement prouvé que ce temps théorique n'est pas réalisable, du moins, avec l'utilisation d'un composant nommé TBC (Test Bus Controller) pour effectuer la transmission des données selon le protocole JTAG. En effet, avec cet élément dans la chaîne de transmission des données de configuration, nous devons tenir compte de certains temps de pause qui lui sont nécessaires pour recharger ses FIFO de données. Le temps complet de reconfiguration d'une unité de traitement se situe alors entre 220 et 230 ms. Même si le temps de reconfiguration optimal n'a pas été atteint, le système conçu montre une réalité de reconfiguration dynamique qui n'a encore jamais été vu à ce jour sur le marché.

ABSTRACT

This master's thesis deals with Run Time Reconfiguration of FPGAs. Run Time Reconfiguration is a relatively new technique that allows to change the hardware configuration of a system while computing. Two distinct paths seem to impose themselves: run time reconfiguration using FPGAs that can be partially reconfigured and run time reconfiguration using conventional FPGAs. These branches are quite different and bring distinct advantages and inconvenients. This research is based on run time reconfiguration using conventional FPGAs, but we have also considered the other run time reconfiguration technique using partially reconfigurable FPGAs.

The prototype system that we have implement is based on a commercially available board called the X-C436, from MiroTech Microsystems Inc of Ville St-Laurent. This company mandated us to implement a run time reconfiguration system based on their product using a secondary communication link supporting IEEE 1149.1 Boundary Scan standard, also known as JTAG. In this project, we implement run time reconfiguration with conventional FPGAs, since the board is equiped with this kind of FPGA, two XC4036EX from Xilinx. We have to adapt this existing board to support RTR by modifying its controller, the CSU. This unit is implemented with another FPGA of the XC3000 family from Xilinx.

Run time reconfiguration on the X-C436 is based on the fact that the board has two processing elements. Thus, we can use one of them to compute while the other is reconfigured. Processing can continue while we reconfigure one of the processing elements. The kind of systems for which that run time reconfiguration can be beneficial is those that can be partitioned in multiple sub-systems. Each sub-system can be implemented, one after the other, in the processing element. The hardware complexity of the system can be significantly reduced by using this technique, since we systematically reuse the same two processing elements of the X-C436 board.

Since the hardware mechanisms were already built, we implemented the software to control it and a demonstration application to validate the system. Since the

board is designed to be used with TMS320C4x DSP from Texas Instruments, the demonstration uses two of these DSPs. This demonstration consists in a flow of several Sobel filters that detect edges with an intensity projected on a rotating vector.

That implementation allowed to measure the reconfiguration time required for each filter. Using the JTAG link, we transmit the configuration file to the board and data is transmitted serially at 8.25 MHz. The complete reconfiguration of a processing element including the time required to switch from one processing element to the other can be, in theory, less than 100 ms. However, measurements showed that this time is not really feasible if we use a component called the TBC (Test Bus Controller) to do data transmission. Indeed, with this component, the transmission chain must pause while the TBC reloads its FIFOs to transmit new data. The complete reconfiguration time was measured to be between 220 and 230 ms. Even if the optimal reconfiguration time has not been reached, the system shows a real run time reconfigurable operation which has no equivalent at this time on the market.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

Liste des Sigles et Abréviations

ASIC	Application Specific Integrated Circuit
C40	Processeur TMS320C40 de Texas Instruments
CCM	Custom Computing Machine
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
CPM	Communication Port Manager
CPS	Connexions par Seconde
CSU	Configuration and Shutdown Unit
DMA	Direct Memory Address
DR	Data Register
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
FIFO	First In First Out
FPGA	Field Programmable Gate-Array
GAL	Generic Array Logic
IR	Instruction Register
JTAG	Join Test Action Group
LUT	Look Up Table
MHz	Méga Hertz
ms	milli seconde
µs	micro seconde
ns	nano seconde
PAL	Programmable Array Logic
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PLA	Programmable Logic-Array
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RISC	Reduce Instruction Set Computer
RTR	Run Time Reconfiguration
SLU	Swappable Logic Unit

TAP	Test Access Port (Contrôleur)
TBC	Test Bus Controller
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TRST	Test Reset
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VPE	Virtual Processing Element

INTRODUCTION

Le monde de la micro-électronique a énormément évolué dans les 50 dernières années. Depuis l'avènement des premiers circuits intégrés dans les années 60, le nombre d'applications découlant de l'utilisation de tels composants n'a cessé de croître. Pour réaliser des circuits complexes, le choix se portait alors soit sur les microprocesseurs à prédominance logicielle, ou encore sur les circuits intégrés dédiés à prédominance matérielle, en plus de disposer d'une myriade de composants élémentaires de type portes logiques de base. Les circuits intégrés dédiés qu'on nomme aussi ASIC pour *Application Specific Integrated Circuit* sont des composants qui ont, certes, beaucoup évolués depuis leur début en terme de capacité, mais qui n'en demeurent pas moins longs et coûteux à concevoir. Les microprocesseurs ont eux aussi évolués. Ils sont en mesure d'accomplir des instructions plus complexes et atteignent des vitesses de traitement toujours plus rapides. Ils conservent cependant un champ d'application restreint, c'est-à-dire strictement logiciel. À l'aide de ces deux ressources principales, soit le microprocesseur et le ASIC, nous avons pu, par le passé, construire des systèmes efficaces, basés soit sur l'un ou sur l'autre, sans toutefois atteindre la rapidité de développement que requiert le marché.

Au milieu des années 80 est né le tout premier FPGA (Field Programmable Gate Array). Ce composant de la logique programmable regroupe un nombre fixe de portes logiques de base qui peuvent ou non être utilisées afin de former un circuit spécifique. Le FPGA se différencie des autres membres de la logique programmable, puisqu'il permet de construire des circuits plus complexes et permet ainsi de servir à autre chose que de la "colle" logique (*glue logic*). En effet, ayant en général plus de ressources matérielles et une architecture plus étoffée, le FPGA permet de concevoir des circuits adaptés à des applications aussi différentes que le contrôle et le traitement de données. Il possède une mer de portes logiques de bas niveau regroupées en cellules qui possèdent chacune au moins un élément de mémoire (bascule D ou *latch*). Ces cellules peuvent être interconnectées entre elles au moyen d'une multitude de réseaux possibles, pré-tracés à l'intérieur du composant.

On catégorise les FPGA par famille selon l'architecture, le type ou la quantité de routage qu'ils offrent et la structure de leurs cellules logiques. Par exemple, certains FPGA avantagent le traitement de données, puisqu'ils ont de grosses cellules logiques et fournissent moins de matrices d'interconnexions. On parle alors d'une grosse granularité. D'autres allouent une plus grande flexibilité, fournissant de petites cellules logiques et une grande quantité de routage. On parle alors de granularité fine et ces types de FPGA répondent mieux aux besoins d'un circuit de contrôle par exemple. Les FPGA ont aussi comme avantage d'être plus rapides que les microprocesseurs dans certaines applications et ils sont définitivement plus flexibles que les circuits intégrés dédiés. Ils peuvent donc rallier le meilleur des deux mondes à moindre coût de développement qu'un circuit intégré dédié.

Plusieurs avantages militent en faveur des FPGA. Le premier est sans doute sa reprogrammabilité. Plusieurs applications utilisent les FPGA soit à titre d'élément de base (traitement) ou à titre de soutien (contrôle). On voit aussi très souvent les FPGA comme coprocesseur à certains systèmes, exécutant une tâche spécifique qui permet de libérer le processeur principal. Les machines de calcul dédiées (traduction de *Custom Computing Machines* ou CCM) utilisent de plus en plus les FPGA à la base du système. La popularité des FPGA fait en sorte qu'on exige plus de rendement, et plus de flexibilité de leur part. On veut à la fois des outils plus performants pour le placement routage, et pouvoir les reprogrammer le plus rapidement possible afin de construire des systèmes de traitement temps réel moins coûteux.

La demande pour une flexibilité accrue a poussé les chercheurs et les industriels vers un type particulier de reprogrammation, qui s'effectue pendant que le système traite. C'est ce qu'on appelle la reconfiguration dynamique, traduction de RTR pour *Run Time Reconfiguration*. En résumé, plusieurs visions du concept sont développées et les voies se séparent entre une reprogrammation complète du FPGA versus une reprogrammation partielle. La reconfiguration dynamique partielle d'un FPGA sous-entend qu'un mécanisme, interne au FPGA, permet de garder une partie de la circuiterie en fonction alors qu'il reçoit des données de reprogrammation pour une région seulement. La reconfiguration dynamique totale, quant à elle, consiste à reconfigurer le FPGA en entier pendant que celui-ci effectue un traitement. Bien

entendu, celui-ci ne doit plus être fonctionnel et un arrêt de traitement doit donc être envisagé.

Les industriels et les universitaires se sont majoritairement tournés vers la reprogrammation dynamique impliquant la reconfiguration partielle du FPGA. On a donc vu apparaître sur le marché des FPGA supportant cette fonctionnalité. Par contre, ces composants n'ont pas remporté un succès commercial significatif. En effet, même si la moitié des FPGA reprogrammables partiellement était issue du milieu académique et n'était destinée qu'à des fins de recherche, les autres composants provenant du milieu industriel n'ont guère fait meilleure figure et n'ont survécu que quatre ans sur le marché. Toutefois, tout récemment, un chef de file dans le marché des FPGA, la compagnie Xilinx, a créé un FPGA pouvant être configuré partiellement, le Virtex. Celui-ci brille davantage pour sa taille et ses capacités d'implantation que pour ses aptitudes à supporter la reconfiguration partielle.

La reconfiguration dynamique utilisant la reprogrammation totale du FPGA a tout de même intéressée quelques chercheurs au tout début des années 90. Ceux-ci ont fourni la preuve que des systèmes dédiés, basés sur des technologies de FPGA conventionnels, pouvaient fournir des résultats remarquables sans inconvénients. En effet, les FPGA reconfigurables partiellement ont causé plusieurs maux de tête aux développeurs, étant donné qu'aucun outil efficace de placement routage et de développement n'était disponible sur le marché pour ces FPGA. Malheureusement, l'intérêt pour ce type de reconfiguration dynamique s'est estompé et, à ce jour, l'ensemble des recherches sur la reconfiguration dynamique est orienté vers les FPGA partiellement reprogrammables.

Ce mémoire contribue à démontrer qu'on peut toujours faire des systèmes performants basés sur la reconfiguration dynamique de FPGA conventionnels. Pour ce faire, nous présenterons d'abord plusieurs études sur le sujet qui ont, d'une manière ou d'une autre, fait évoluer les connaissances sur le sujet. Nous ferons ressortir les lacunes visibles des systèmes utilisant les FPGA reconfigurables partiellement et nous fixerons, de cette manière, les objectifs de notre projet. Le concept et les notions de bases relatifs à ce projet seront présentés au chapitre deux, tandis que l'implantation en tant que telle sera le thème du chapitre trois. De plus, à la fin de ce mémoire, nous

ferons une analyse critique de ce projet, des manières de l'améliorer et de comment nous pouvons l'étendre à des systèmes plus performants.

CHAPITRE PREMIER - REVUE DE LITTÉRATURE

1.0 INTRODUCTION

Ce chapitre vise à familiariser le lecteur au sujet traité dans le cadre de ce mémoire en le situant parmi les recherches antérieures et actuelles du milieu académique et industriel. Cette revue de littérature se veut exhaustive afin de favoriser chez le lecteur une compréhension plus affinée pour les chapitres subséquents. Après une brève description des tendances actuelles, une discussion visant à faire ressortir les points forts et les points faibles de ces recherches sera faite. Finalement, nous présenterons le sujet de cette recherche en démontrant quels types de problèmes nous tentons de résoudre.

Le milieu académique et le milieu industriel ont présenté des recherches et des produits ayant trait à la reconfiguration dynamique des FPGA depuis les années 90 environ. Dans le but de faciliter la présentation et la compréhension de ces recherches, nous les regrouperons en différents thèmes. Tout d'abord, nous présenterons les recherches qui traitent de la reconfiguration dynamique des FPGA supportant la reconfiguration partielle (RTR partiel) et la reconfiguration dynamique des systèmes comportant des FPGA conventionnel (RTR total). Ensuite, nous présenterons des recherches qui se penchent sur des outils de placement et routage ou des outils de développement pour des systèmes supportant le RTR. Nous exposerons également des recherches qui ont mené au développement de nouvelles architectures de FPGA qui supporteraient également une forme de reconfiguration dynamique. Finalement, nous présenterons les quelques recherches qui ont fait une analyse du RTR au point de vue rendement et avenir.

1.1 RECONFIGURATION DYNAMIQUE DE FPGA POUVANT ÊTRE RECONFIGURÉS PARTIELLEMENT

La reconfiguration dynamique de FPGA qui supportent la reconfiguration partielle est sans contredit le sujet le plus répandu des recherches portant sur le RTR. En effet, avec l'avènement sur le marché de FPGA ayant de telles capacités, beaucoup y ont vu la solution idéale pour plusieurs applications. Quatre compagnies principalement ont produit pour le marché ou pour le milieu académique des FPGA pouvant être reconfigurés partiellement. Déjà vers la fin des années 80 (1988), Algotronix, une compagnie d'Écosse, présente une première version de son FPGA CAL1024, qui supporte la reconfiguration partielle. Cette entreprise est, vers la fin de 1995, achetée par Xilinx, qui en fait la base de sa famille XC6200. Le FPGA CLAy de National SemiConductor a, quant à lui, toujours servi à des fins académiques et a alimenté plusieurs des recherches présentées plus loin. Finalement, la famille 6000 d'Atmel, disponible commercialement, a été utilisée dans quelques recherches. À ce jour, les produits d'Atmel et de Xilinx sont encore disponibles, par contre, les XC6200 de Xilinx ne sont plus supportés. Xilinx a plutôt misé sur une toute nouvelle famille de FPGA qui offre, en plus d'une capacité d'implantation de circuits de grande taille, la possibilité de supporter la reconfiguration dynamique en utilisant un processus de reconfiguration par colonne du FPGA. Cependant, puisque cette toute nouvelle fonction de reconfiguration partielle n'est annoncée publiquement que depuis le milieu de l'été 1999, nous n'avons pu étudier ce nouvel aspect dans le cadre de ce mémoire. Nous y consacrons toutefois une attention particulière au chapitre quatre. Mentionnons finalement qu'une compagnie anglaise du nom de Plessey SemiConductors avait commercialisé durant un bref temps des FPGA de ce type au tout début des années 90, mais nous n'avons retracé aucune recherche les ayant utilisés.

1.1.1 XILINX XC6200 OU CAL D'ALGOTRONIX

Les recherches qui sont présentées dans cette section concernent des applications ayant été réalisées avec les FPGA de la famille XC6200 de Xilinx ou CAL

d'Algotronix. En effet, plusieurs outils de placement/routage, ainsi que des outils de développement de systèmes RTR à base de ces FPGA ont été réalisés, mais nous en discuterons à la partie 1.3 qui traite des outils.

Même si les FPGA d'Algotronix sont apparus sur le marché en 1988, ce n'est pas avant 1993 qu'un article relatant l'utilisation de ce FPGA est publié (Foulk, 1993). Le système traité dans cet article a trait à une technique nommée data-folding (pliage de données) spécialement appliquée à des recherches à l'intérieur de textes. Le FPGA d'Algotronix est, selon l'auteur, idéal pour la reconfiguration dynamique, puisqu'il possède une fine granularité, c'est-à-dire que de très petites fonctions logiques (2 entrées) sont garnies d'une excellente structure de routage interne. L'auteur mentionne qu'il est possible de sauver plus de la moitié de l'espace prévu pour implanter le circuit. Ainsi, son circuit ne nécessite-t-il que deux FPGA CAL d'Algotronix à la place de cinq. Finalement, l'auteur traite d'un aspect qui est rarement abordé dans les recherches sur le sujet, soit l'implantation d'un système pour commuter d'une configuration à l'autre. Dans ce cas précis, l'auteur mentionne que le FPGA CAL d'Algotronix offre la possibilité de relire l'état de chacune des cellules. Générant une petite logique de vérification, il est en mesure de connaître en tout temps le moment adéquat pour la commutation. Le mécanisme est automatique. De cette manière, on peut augmenter la vitesse de commutation du système.

Le tout premier article répertorié qui traite de la reconfiguration partielle avec les XC6200 de Xilinx est l'article de Brebner et Gray (1995), qui utilise ces FPGA pour la détection de codes de longueur variable à la rapidité d'un signal vidéo (temps réel). Ces codes sont fournis selon la norme fax T.4 et le codage des symboles est fait selon un codage de Huffman. Ce type de codage fait en sorte qu'on peut encoder les symboles à transmettre selon leur fréquence d'apparition. Les symboles les plus souvent lus possèdent un code binaire court (au minimum 2 bits) et les symboles les moins souvent rencontrés ont un code binaire plus long (maximum 13 bits). Le signal à décoder est sériel et passe à travers un arbre de décodage qui peut décoder les symboles qui ont un maximum de 6 bits. Au-delà de ce nombre, le FPGA doit être reconfiguré partiellement en sélectionnant un parmi les 64 sous-arbres afin d'être en mesure de traiter les symboles de plus de 6 bits. La reconfiguration de ce sous-arbre de décodage nécessite 10 cycles d'écriture (dû à leur architecture de système). Mais

compte tenu que 90 % des symboles possèdent un code plus petit ou égal à 6 bits, le temps de reconfiguration nécessaire lors des codes plus long peut être amorti et on peut compter une moyenne d'un cycle d'écriture par mot binaire (peu importe sa longueur). Donc, d'après leurs calculs, l'équipe comptait obtenir une sortie traitée qui rencontrait la bande passante vidéo. Étant donné que les FPGA XC6200 n'étaient pas disponibles sur le marché au moment où les travaux ont été effectués, les auteurs n'ont pu implanter cette application.

L'année suivante (1996), Gunther, Milne et Narasimhan publient sur une application faite à partir d'un système de 16 FPGA CAL d'Algotronix. Ce groupe étudie la recherche de mots à l'intérieur de textes. Une liste contenant une série de mots choisis devient la base de la comparaison pour chacun des mots du texte. À la fin du texte, celui-ci obtient un score qui est proportionnel au nombre de mots correspondants à la liste. Le mécanisme de recherche est composé d'un circuit fixe et d'un chemin de données reconfigurable. Cette logique, qui utilise 57% de l'espace total du système de 16 FPGA, est synthétisée juste avant d'être configurée. Selon leurs résultats, ils sont en mesure d'effectuer une recherche de 91 mots-clés en 3.21 secondes, ce qui inclut 0.16 seconde pour la synthèse des chemins de données et 3.05 secondes pour leur configuration à l'intérieur du système de FPGA. Mentionnons également que la recherche s'est effectuée sur plusieurs textes totalisant une base de données de dizaines de méga-octets.

Une équipe de l'Université de Glasgow en Écosse, Royaume-Uni (Burns et al., 1997) s'est intéressée à la reconfiguration dynamique de systèmes à base de FPGA partiellement reconfigurables. Le but de leur recherche était d'établir une interface pour un système exécutant une application dynamiquement reconfigurable. Pour ce faire, ils ont d'abord étudié les besoins de tels systèmes en démontrant trois applications. De celles-ci, ils ont pu caractériser le système à bâtir. Le système est conçu pour utiliser un FPGA de la famille XC6200, mais le concept pourrait aussi être implanté sur d'autres FPGA partiellement reprogrammables. Leur système nommé RAGE contient quatre modules principaux, en plus de la banque d'applications et du FPGA lui-même. Ces modules sont le *Virtual Hardware Manager*, le module *Transformation Manager*, le module *Configuration Manager* et finalement le module *Device Driver*. Le premier est le contrôleur du système qui interagit avec tous les

modules sauf le FPGA. Le second module, *Transformation Manager*, est un outil qui permet de réorganiser les parties à reconfigurer du FPGA pour que celui-ci puisse demeurer performant d'une configuration à l'autre. Le module *Configuration Manager* configure le FPGA par l'entremise du *Device Driver*, tandis que ce dernier s'occupe du transfert de données/résultats, ainsi que des données de reconfiguration. Sur la base de ce que l'on retrouve dans la littérature, le projet semble en être resté au stade d'une proposition non développée.

Dans un autre ordre d'idée, une équipe de l'Université du Massachusetts à Amherst (Park et Burleson, 1998) a développé une technique qui utilise les FPGA partiellement reconfigurables pour une application dédiée à l'estimation de mouvements. Cette technique de compression d'images est, selon les auteurs, la plus exigeante au niveau temps de calcul parmi toutes les techniques utilisées dans le codage vidéo. La quantité de CLB à reconfigurer est autour de 64. De ce nombre, ils concluent que le temps de reconfiguration de leur système sera autour de 20 micro-secondes, étant donné que d'après les spécifications du composant, le temps de reconfiguration d'un CLB est autour de 40 nano secondes. Ceci est amplement suffisant pour du traitement temps réel à raison de trente images par seconde, 2 champs par image, un pair et un impair. On ne parle pas de l'implantation physique de cette recherche.

Étant donné leur grande disponibilité, les FPGA de la famille XC6200 de Xilinx ont été amplement cités et utilisés dans le milieu académique. Comme il est impossible de relater tous les travaux de recherche ayant utilisé de tels FPGA, nous nous sommes limités aux recherches qui sont les plus importantes dans le contexte de ce mémoire.

1.1.2 CLAY DE NATIONAL SEMICONDUCTOR

Le FPGA CLAy de National SemiConductor a essentiellement été utilisé par le milieu académique à des fins de développement. Les premières recherches qui traitent de ce FPGA datent de 1995.

Un des plus importants systèmes ayant été construits à partir des FPGA CLAy de National SemiConductor est sans doute le DISC de L'Université Brigham Young,

Utah par Wirthlin et Hutchings (1995). Le système reflète l'essence même de toute application dite dynamiquement reconfigurable, c'est-à-dire qu'une certaine logique de base est fixe et qu'une instruction à la fois est portée à l'intérieur du FPGA selon le cours de l'exécution du programme. Cette instruction fait partie d'une suite d'instructions pré-compilées qui constitue un module spécifique dans une bibliothèque d'instructions. Le système qui entoure DISC interagit avec un ordinateur hôte via un bus ISA. Par la suite, deux FPGA CLAy31 sont utilisés, le premier est le processeur DISC qui reçoit toutes les instructions et le deuxième agit à titre de contrôleur pour exécuter la configuration. Le processeur a également à sa disposition de la mémoire RAM. Jusqu'à cinq instructions peuvent être présentes à la fois dans le processeur, mais une à la fois est exécutée. Lorsque le processeur contient cinq instructions et qu'une autre instruction doit être configurée, la plus ancienne est effacée. Ainsi, le processeur agit à titre de cache des cinq plus récentes instructions.

Les suites de ce projet apparaissent dans un autre article de Wirthlin et Hutchings l'année suivante avec DISC II. La seconde version de ce système est un peu plus élaborée que la première avec plus de mémoire, plus de registres et un FPGA de plus. La logique fixe est moindre, afin de donner plus de place à la partie dynamique. Le FPGA supplémentaire a hérité de la partie instruction, le processeur garde seulement le programme épuré. La mémoire est accessible par le processeur et par le FPGA contenant les instructions. Le processeur fonctionne maintenant sur 16 bits au lieu de 8 et à une vitesse de 8 MHz au lieu de 7.5.

Une autre équipe de l'Université Brigham Young, Hadley et Hutchings (1995) a utilisé les FPGA CLAy de National SemiConductor pour construire un réseau de neurones artificiel. Le système RRANN2 pour *Rapidly Reconfigurable Artificial Neural Network*, tire sa source d'études antérieures qui exploitaient des FPGA conventionnels (XC3010 de Xilinx). Les réseaux de neurones auraient avantage à utiliser la reconfiguration dynamique, puisqu'elle peut augmenter la densité fonctionnelle du réseau de 500% à comparer des réseaux implantés sur des systèmes n'utilisait pas la reconfiguration dynamique. Le but du projet RRANN2 est d'atteindre les mêmes performances que son prédécesseur, mais avec encore moins de neurones. De plus, l'équipe voulait approfondir le développement d'une méthodologie de conception avec les FPGA partiellement reconfigurables. En effet, en maximisant la taille du circuit

statique et en minimisant la partie dynamique, le temps de reconfiguration en sera d'autant diminué. Les auteurs mentionnent finalement qu'il existe un manque important au niveau des outils de synthèse et de simulation qui tiennent compte de tous les mécanismes impliqués dans le développement de systèmes à base de FPGA partiellement reconfigurables.

Au niveau application concrète, Schoner, Jones et Villasenor présentaient en 1995 un système de codage vidéo sans fils utilisant des FPGA reconfigurables dynamiquement. L'application utilise trois configurations consécutives qui conservent une grande partie du circuit de base de l'une à l'autre. Cette suite de configurations est effectuée pour chaque champs d'images. Le temps total de ces reconfigurations est de 3 ms, ce qui correspond à 10 % du temps d'un cadre. À l'écriture de l'article, la réalisation du projet n'était pas terminée.

1.1.3 ATMEL 6000

La famille de FPGA Atmel 6000 était autrefois offerte sous la bannière de Concurrent Logic et elle portait le nom de Cli6000. Les recherches faisant état de l'utilisation de ces FPGA ne sont pas nombreuses: quelques unes en 93-94 et une en 1996. Le manque d'intérêt du milieu académique est curieux, puisque Atmel est à ce jour la seule compagnie qui fournit sur le marché des FPGA programmables partiellement. Un FPGA plus récent, le AT40k, est lui aussi reprogrammable partiellement. Cette famille offre de 5k à 50k portes logiques utilisables. On peut donc se questionner sur la qualité du support de ces FPGA, car un manque d'outils veut dire immédiatement perte d'intérêt dans le domaine du FPGA.

Ross, Vellacott et Turner (1993) ont développé un système dédié au traitement d'images. Leur système nommé *imputer* pour **im**aging - com**puter** utilise des FPGA Atmel 6000 comme unités de traitement, mais un FPGA conventionnel (Actel) comme contrôleur. Une mémoire RAM est accessible par ces deux composants pour des fins de traitement et une autre mémoire est utilisée à titre de bibliothèque des configurations à être implantées. Des résultats sur des traitements d'images comme un filtre Sobel sont présentés à titre de comparaison avec un système compétiteur. Selon les auteurs, leur système serait jusqu'à dix fois plus rapide avec cette implantation de

reconfiguration dynamique. Malheureusement, aucune donnée relative à la taille de l'image traitée n'est fournie, ce qui nous empêche de véritablement évaluer les performances de ce système.

En 1993 et 1994, deux groupes de recherche ont présenté une étude d'implantation d'un réseau de neurones artificiel sur les FPGA de Atmel (Concurrent Logic pour l'un d'eux). Le premier groupe dirigé par van Daalen (van Daalen, Jeavons et Shawe-Taylor, 1993) de l'Université Royal Holloway de Londres n'a pas implanté concrètement son architecture sur ces FPGA. Par contre, tout le projet a été pensé en conséquence de cette implantation. Le deuxième groupe, de l'Université de Strathclyde (Lysaght et al., 1994), donne plus de détails quant à l'implantation et aux résultats. En effet, pour leur architecture, le réseau de neurones implanté sur un système dynamiquement reconfigurable obtient 38 fois les performances qu'ils auraient obtenues avec un système basé sur des FPGA conventionnels. Leur système offre une performance de 770K CPS (Connexions par seconde) à 24KHz comparativement à 20K CPS, 625 Hz si la même architecture avait utilisé des FPGA conventionnels. Par contre, l'équipe mentionne que de bien meilleurs résultats avaient été obtenus avec des systèmes plus volumineux basés sur des FPGA conventionnels. Ils nomment GANGLION, un projet de Cox et Blanz (1992) qui utilise 30 FPGA XC3010 de Xilinx. Ces circuits sont relativement plus gros que les AT6005 utilisés dans le projet de Lysaght. Les performances de GANGLION sont impressionnantes, 4.5 milliards de connexions par seconde (4.45G CPS).

Finalement, une dernière étude utilisant les FPGA de Atmel a été publiée, celle d'Eggers et al., (1996). Cette fois-ci, l'application touche un système rapidement reconfigurable de commutation croisée (Crossbar Switching). Cette application, destinée au domaine biomédical, avait comme contrainte une reconfiguration totale du système en deçà de 200 micro secondes. Le FPGA 6005 de Atmel peut se configurer totalement en 800 micro secondes, en utilisant le mode le plus rapide de reconfiguration. La reconfiguration dynamique utilisant le procédé de reconfiguration partielle ne peut donc pas s'effectuer aussi rapidement. En effet, si l'on considère que la reconfiguration complète du composant prend environ 255 nano secondes par cellule (3136 cellules individuelles), la reconfiguration partielle est environ 6 à 7 fois plus lente avec un temps de 1.7 micro secondes par cellule. Le temps requis par cellule

pour entrer dans la contrainte de 200 micro secondes est de 3.125 micro secondes par cellule. L'implantation n'atteint toutefois pas les performances d'un système dédié à cette application (FPID de I-Cube). Par contre, en considérant le rapport performance sur prix, la conception sur FPGA reprogrammable est la plus avantageuse. L'équipe de recherche note en terminant que les FPGA de Xilinx, plus récents, promettent d'être plus performants dans le sens où l'on peut les reconfigurer plus rapidement que les FPGA d'Atmel en mode reconfiguration partielle.

1.2 RECONFIGURATION DYNAMIQUE UTILISANT DES FPGA CONVENTIONNELS

Peu de recherches relatent l'utilisation de FPGA conventionnels pour concevoir un système pouvant être reconfiguré dynamiquement. Les recherches qui ont toutefois été publiées ont été faites au tout début des études portant sur la reconfiguration dynamique de systèmes à base de FPGA.

Le tout premier article traitant de la reconfiguration dynamique de systèmes de FPGA conventionnels apparaît en 1991. Lysaght, de l'Université de Strathclyde à Glasgow, Écosse, Royaume-Uni, fait état d'une architecture développée dans le cadre de projets de fin d'études de deux étudiants en génie électrique de l'Université. Ces deux projets, qui se soutiennent mutuellement, proposent un système simple avec une application simple. Deux FPGA de la famille XC2000 de Xilinx interagissent pour former un système qui dirige sa propre reconfiguration. En fait, un seul des FPGA exécute toute l'application tandis que l'autre agit à titre de contrôleur. C'est pourquoi, selon leur application, pour passer d'une configuration à une autre, on doit attendre le temps de reconfiguration total du FPGA traitant. Ce temps d'arrêt est court, puisque les FPGA XC2000 sont petits, mais l'auteur mentionne tout de même que l'utilisation des FPGA reprogrammables partiellement pourrait être envisagée, puisqu'en configurant le système partiellement, le temps de reconfiguration est diminué.

Un impressionnant système de 30 FPGA XC3090 de Xilinx est présenté en 1992. Il s'agit de GANGLION, de Cox et Blanz dont nous avons parlé auparavant (section 1.1.3). Cet article démontre l'implantation d'un classificateur de connexions ou

encore réseau de neurones. Ces chercheurs ont développé un concept intéressant qui consiste à générer de la logique spécifique à chaque nouvelle application. Ainsi, à titre d'exemple, ils ont implanté un multiplieur de 8 bits variables par 8 bits constants. Les 8 bits constants sont fixés à l'intérieur de LUT (Look Up Table), ce qui permet ainsi d'avoir un multiplieur consommant moins de CLB, en plus d'être plus rapide qu'un multiplieur conventionnel. Le système en lui-même est très performant. Comme nous l'avons déjà mentionné, il est en mesure d'effectuer 4.48G connexions par seconde et 20 millions de décisions par seconde. Selon les auteurs, leur système utilise pleinement toutes les capacités de la reprogrammabilité des FPGA conventionnels.

En continuant avec les réseaux de neurones, Eldredge et Hutchings (1994) ont publié deux articles relatifs à une implantation sur des FPGA XC3090 de Xilinx. Le projet RRANN pour Run-time Reconfiguration Artificial Neural Network procède à l'implantation d'un algorithme dit de *backpropagation*. Pour ce faire, ils divisent le processus en trois parties distinctes qui sont tour à tour implantées dans le même FPGA. Basé sur la propagation d'erreur, le système effectue les mêmes trois étapes (i.e. calcul, propagation de l'erreur, rafraîchissement) jusqu'à ce que le réseau soit suffisamment entraîné. Le système a été monté avec 12 FPGA, dont un dédié à être le contrôleur et les autres supportent chacun six neurones matériels. Comparativement à un système n'utilisant pas de reconfiguration dynamique, la densité de neurones implantés serait augmentée de 500%. Par contre, il est bon de mentionner que le système utilisant la reconfiguration dynamique surpasse le système qui ne l'utilise pas, lorsque le nombre de FPGA XC3090 est plus grand que 23. Nous pouvons facilement comparer ce système avec GANGLION présenté précédemment, puisque celui-ci utilisait également des FPGA XC3090 de Xilinx. En vérifiant les résultats fournis par les auteurs dans une charte de leur article, avec 30 FPGA XC3090, leur système serait en mesure de fournir environ 110M CPS (1.1×10^8 connexions par seconde) alors que le système GANGLION avait comme performance 4.48G CPS soit 40 fois mieux.

Une dernière application présentée en 1995 par Lemoine et Merceron utilise une plate-forme DECPeRLe-1 du laboratoire de recherche de Digital à Paris. Ils ont implanté un système pour traiter une base de données contenant tout le génome humain. La carte DECPeRLe-1 utilise 23 FPGA XC3090 de Xilinx et permet

l'implantation d'un système utilisant la reconfiguration dynamique des FPGA. Deux types de reconfiguration dynamique sont envisageables selon les auteurs. La première synthétiserait un fichier de configuration totalement nouveau, conçu pour satisfaire des contraintes connues durant le traitement. La deuxième modifierait un fichier de configuration de base selon des informations connues durant le traitement. Il est clair que générer un fichier de configuration totalement nouveau pendant que le système traite ralentit celui-ci d'un délai considérable. Cependant, selon les auteurs, ce processus permettrait d'augmenter la vitesse de traitement du FPGA, ainsi que sa densité. Cette équipe a démontré que le traitement de 3.5 milliards de nucléotides pouvait s'effectuer en 100 secondes environ. Ils ont comparé ce temps avec celui d'un processeur RISC 3000 40MHz (même vitesse que leur système) et ils ont obtenu un temps dans les dizaines de millier de secondes.

1.3 Les outils

La quantité de recherches sur le développement d'outils de placement/routage, ou d'outils de développement et de simulation pour les FPGA partiellement reprogrammables, en dit long sur la volonté du milieu académique de travailler sur de tels FPGA et l'intérêt que l'industrie y voit. Clairement, pour construire un système reprogrammable dynamiquement, on a besoin d'outils de placement et de routage, ainsi que d'outils de simulation adéquats. Ces outils, les fabricants de FPGA les ont et ils sont généralement suffisamment performants pour répondre aux besoins des utilisateurs, quoiqu'ils ne soient pas développés pour supporter des applications utilisant la reconfiguration dynamique. Par contre, lorsque les FPGA partiellement reprogrammables sont impliqués, les choses se compliquent. Les articles dont nous discuterons dans cette section ont aussi et surtout fait état d'outils de niveau système qui supportent tout le processus de développement de circuits dynamiquement reconfigurables.

Un des premiers articles traitant de la réalisation d'outils liés au développement des systèmes dynamiquement reconfigurables est celui d'un groupe logiciel du David Sarnoff Research Center au New Jersey (Gokhale et Marks, 1995). Cet outil vise à

partitionner un programme parallèle afin de l'implanter dans un processus de reconfiguration dynamique. L'outil est développé pour les FPGA CLAy de National Semiconductor qui sont partiellement reconfigurables. Dans le processus de partitionnement, le compilateur conserve les "variables globales" (sic) entre chaque reconfiguration, ce qui forme la circuiterie fixe. Leur compilateur génère à la fois le VHDL à être synthétisé par les outils conventionnels et le code C qui exécutera l'application sur le système hôte. Au moment où cet article était en rédaction, le travail à effectuer sur le compilateur n'était pas encore terminé.

Un auteur qui a constamment travaillé sur des outils de reconfiguration dynamique est Brebner du département d'informatique de l'Université d'Edinburgh en Écosse, Royaume-Uni. Depuis 1996, Brebner raffine les outils qui supporteraient son concept de SLU (Swappable Logic Unit). Le FPGA qui est jusqu'alors visé est le XC6200 de Xilinx. Le concept de SLU repose sur l'identification de fonctions spécifiques, localisées à un endroit fixe et utilisant un certain nombre d'entrées/sorties définies. La manipulation de SLU est la base de leur outil logiciel. Trois articles de 1996 et 1997 exposent le concept sous tous ses angles et l'auteur promet des résultats quant à son implantation. En 1998, Brebner et Donlin s'attaquent au problème de routage, mais sans toutefois montrer de résultats concrets, qu'ils promettent encore pour des travaux futurs. De plus, l'outil qu'ils prévoient développer est toujours destiné au FPGA XC6200 de Xilinx et ils mentionnent même que l'outil est dépendant de la famille de FPGA utilisé, donc non portable. Rappelons également qu'à cette date, les FPGA XC6200 de Xilinx n'étaient plus disponibles commercialement mais placés à la disposition du milieu académique seulement.

Une équipe de l'Imperial College à Londres (Luk, Shirazi et Cheung, 1996) a développé un modèle d'outil pouvant aider à tous les processus de conception d'un système basé sur la reconfiguration dynamique de FPGA. À partir des spécifications, l'outil permettrait de visualiser un système en cours de développement. À cette date, seulement le modèle était développé, l'imbrication de l'aspect temporel entre autre était un manque important. En 1997, l'équipe fournit un compilateur qui est basé sur le langage formel RUBY et le VHDL. L'outil est compatible avec les outils de simulation et de synthèse de l'industrie. L'insertion de la notation formelle RUBY apporte une base pour la vérification formelle et la validation, notions qui n'ont pas encore été exploitées

dans la communauté. Même si l'outil n'est pas développé pour une famille de FPGA en particulier, les FPGA XC6200 de Xilinx ont servi de plate-forme d'essai pour deux exemples.

La même équipe (Shirazi, Luk et Cheung) publie en 1998 un outil un peu plus substantiel, qui introduit le concept d'identification et de planification de région reconfigurable à l'intérieur du composant visé. Le principal objectif de l'outil est qu'il soit en mesure de reconnaitre la partie commune à deux circuits distincts dans le but de les implanter consécutivement. Cette partie fait toujours allusion au tout premier plan de travail que l'équipe avait démontré en 1996. Bizarrement, l'outil présenté dans cet article concerne la toute première étape de leur plan de travail, qui comprend les étapes suivantes; décomposition du circuit, séquencement, évaluation partielle, calcul de configuration incrémentielle, génération de configuration simultanée et validation. La représentation des deux configurations selon un graphe bipartie permet de décomposer les deux circuits et d'en faire ressortir, selon un système de poids, les parties semblables entre les deux. En utilisant les meilleurs résultats de cette analyse, la position de l'insertion de multiplexeurs est identifiée et les deux circuits sont ainsi combinés. Les auteurs ont démontré pour quelques applications une évaluation du nombre de cellules ainsi que du nombre de cycles de reconfiguration nécessaires pour la réalisation du circuit. Jusqu'à 32 cycles de reconfiguration sont nécessaires pour l'implantation d'un *pattern matcher*. La méthode proposée permet de réduire le nombre de cellules implantées de moitié et d'augmenter sa rapidité du double. Pour ce qui est de la notion de rapidité, le terme est mal défini. Les auteurs ne mentionnent pas le temps requis pour une reconfiguration, ils mentionnent que le temps de reconfiguration est plus long que le temps de traitement ce qui peut être un désavantage de l'utilisation d'une telle méthode.

Une application intéressante a été produite des suites de ces recherches de l'Imperial College de Londres. En effet, Luk et al. ont publié récemment (FCCM99) un article portant sur l'utilisation de leur outil de développement pour une application vidéo. La réalité augmentée (Augmented Reality) est une technique qui permet de juxtaposer des images réelles et des images synthétiques, pour enrichir la perception de l'environnement ainsi que l'interaction avec le milieu.

Une autre équipe qui provient cette fois de l'Université de Strathclyde à Glasgow, Écosse, Royaume-Uni (Lysaght et Stockwood, 1996) publie le premier outil de simulation et de développement destiné aux systèmes reconfigurables dynamiquement. Comme le soulignent les auteurs, une partie importante des outils reste encore à développer, soit ceux pour l'estimation du temps de reconfiguration, le placement routage automatique, la planification de l'espace (*floorplanning*) et la synthèse du contrôleur de reconfiguration. Toutefois, leur approche est intéressante pour la simulation des systèmes dynamiquement reconfigurables, puisqu'elle permet d'introduire la simulation beaucoup plus tôt dans le processus de développement, en utilisant des approximations de l'effet de la reconfiguration dynamique sur le système.

En 1998, Robinson, McGregor et Lysaght publient une suite du développement de l'outil décrit par la dernière équipe. Cette équipe, qui est également de l'Université de Strathclyde, a orienté sa recherche dans le développement d'un outil complet et plus étoffé, qui aide tout au long du processus de développement, le concepteur de systèmes dynamiquement reconfigurables. Leur modèle semble bien défini, basé essentiellement sur la très importante étape de simulation, ils présentent des chemins et des évaluations bien distincts pour la partie simulation fonctionnelle et simulation avec délais. Une fois les simulations terminées, un de leurs outils nommé DCSTech entreprend la tâche de décomposer le circuit reconfigurable dynamiquement en une série de composants statiques. Les signaux communs sont fixes, ce qui facilite le routage mais contraint le placement. L'implantation physique de ce système n'est toutefois pas encore opérationnelle. En effet, une partie très importante qui concerne le contrôleur de configuration n'est pas encore réalisée. Un autre manque relaté par l'équipe est un estimateur de temps de reconfiguration pour évaluer les latences dans le traitement ou tout simplement pour évaluer si le circuit possède les performances attendues lorsqu'implanté sur un système dynamiquement reconfigurable.

Toujours en Écosse, mais à l'Université de Glasgow cette fois, des chercheurs (Singh, Hogg et McAuley, 1996) ont présenté un outil presque totalement basé sur les méthodes formelles. C'est plutôt sous forme d'étude que les chercheurs nous présentent leur méthode de reconfiguration qui serait basée sur l'évaluation partielle du système. Un exemple simple qu'ont démontré les auteurs pour illustrer l'évaluation partielle est le suivant:

Soit la fonction suivante:

(A et B) ou C.

Si l'évaluation partielle du système permet de définir par exemple que A vaut 1, alors l'expression définie plus haut se résume maintenant à:

B ou C.

La réalisation d'une évaluation partielle sur le système a permis d'éliminer une porte logique, ce qui réduit la complexité matérielle tout en augmentant la rapidité, puisqu'il y a moins de portes logiques à parcourir. Sachant que cette reconfiguration partielle est variable selon le circuit à évaluer, dans quelle situation est-il vraiment avantageux d'utiliser une telle méthode? Étant donné que les auteurs n'avaient pas implanté leur système au moment de la rédaction de l'article, cette question restera probablement en suspend.

Tout récemment, à la dernière conférence FPGA for Custom Computing Machines (avril 1999, Napa Valley, Californie, Etats-Unis), la presque totalité des articles sur la reconfiguration dynamique de FPGA concernait des outils de développement pour de tels systèmes. Certains étaient conçus expressément pour les FPGA de Xilinx XC6200, tandis que d'autres étaient plus génériques. Notons que depuis près d'un an, Xilinx ne supporte plus les FPGA de la famille XC6200.

Nous n'aborderons pas en détail les recherches portant sur le développement d'outils pour les FPGA de la famille XC6200 de Xilinx pour les raisons nommées plus haut. Nous présenterons donc brièvement trois recherches présentées lors de la conférence FPGA for Custom Computing Machines d'avril dernier. La plus originale de ces recherches est probablement celle de Vasilko et Cabanis qui introduit un nouveau type de données dans la bibliothèque déjà définie de IEEE (std_logic_1164). Ce nouveau type de données, "V" pour *Virtual*, permettrait d'indiquer la partie de logique reconfigurée dynamiquement et propagerait la valeur de ce signal aux composants touchés.

Une autre étude portant sur un outil de développement pour les XC6200 de Xilinx est présentée par Cardoso et Neto à la même conférence. Cette équipe de l'INESC du Portugal a développé un outil basé sur le langage Java. Leur compilateur serait en

mesure de décortiquer toute l'information nécessaire à la synthèse matérielle d'un système dynamiquement reconfigurable. Selon la taille du circuit, le système choisi le FPGA dans lequel il sera implanté parmi ceux disponibles de la famille XC6200 de Xilinx.

Poursuivant probablement les travaux entrepris plus tôt à l'Université de Glasgow en Écosse, McKay et Singh ont publié à la même conférence un article portant sur les techniques de déverminage pour les systèmes dont la partie matérielle est dynamiquement reconfigurable. On se souviendra que ce groupe avait poussé leurs recherches vers l'implantation d'un système basé sur l'évaluation partielle du circuit. Ils ont donc jumelé leurs outils à ceux de Xilinx supportant la famille XC6200 soit XACTstep.

Dans un sens plus général, l'équipe de l'Université Brigham Young en Utah (Hutchings et al, 1999), a poursuivi depuis 1998 des travaux en vue de concevoir un outil de développement conçu originellement pour les systèmes dynamiquement reconfigurables. JHDL, pour Java Hardware Description Language, a été porté pour supporter les familles de FPGA non dynamiquement reconfigurables. À la base, (1998), l'outil possédait des constructeurs et des destructeurs d'objets pour décrire les structures de circuits qui sont dynamiquement reconfigurables. Il pouvait également fournir un environnement de simulation et d'exécution, dans lequel l'usager pouvait passer directement de l'environnement logiciel de simulation au domaine matériel permettant l'exécution. De plus, un support particulier est fourni pour les applications de type logiciel/matériel intégré. Une interface logicielle permet à l'usager de visualiser à la fois, le circuit développé, une simulation et l'état des signaux internes. Selon les auteurs, la bibliothèque serait à ce jour plus développée pour les FPGA de la famille 4000 de Xilinx qu'elle ne l'était pour la famille 6000.

1.4 DÉVELOPPEMENT DE NOUVEAUX FPGA OU DE SYSTÈMES DYNAMIQUEMENT RECONFIGURABLES

Le développement de nouvelles architectures de FPGA supportant la reconfiguration dynamique est apparu très tôt dans les recherches sur le sujet. En effet, le manque de support des quelques FPGA existant qui supportaient la reconfiguration dynamique ainsi que l'architecture même des FPGA disponibles ont poussé les chercheurs vers la création de FPGA faits "sur mesure". Les résultats sont très diversifiés et intéressants, mais malheureusement, aucune de ces recherches n'a abouti à une implantation réelle avec mise en marché.

L'équipe de Ling et Amano (1993 et 1995) a été l'une des premières à discuter du principe de *Virtual Hardware* (matériel virtuel) et de l'implanter sur un système conçu pour cette fin. Leur argument principal justifiant la conception d'un nouveau composant est que les FPGA actuels demandent trop de temps pour effectuer une reconfiguration complète. Cette reconfiguration s'effectue habituellement à partir d'une ROM externe au FPGA. C'est pourquoi en créant un FPGA qui possède un certain nombre de configurations à l'intérieur du même composant, le temps de reconfiguration en serait diminué d'autant. À l'aide seulement de multiplexeurs, les multiples configurations pourraient permuter d'une à l'autre en très peu de temps. Le composant, nommé WASMII, pourrait même faire partie d'un réseau de composants du même type et ainsi créer un système de taille aussi grande que voulue. C'est essentiellement pour du traitement de chemin de données que les concepteurs ont pensé leur puce.

Une équipe du Virginia Polytechnic Institute (Bittner et al., 1996) a proposé une machine dédiée au traitement de flot de données basé sur les concepts disponibles avec les technologies de FPGA. Leur concept ne se concentre pas seulement au niveau composant, mais il supporte aussi le niveau système comprenant toute la partie développement et logicielle. Les applications visées avec ce système sont le traitement numérique, les applications DSP (Digital Signal Processing), le traitement d'images, les systèmes de communication, les filtres numériques, le traitement de sons, le contrôle de système temps réel et l'accélération de simulation. Les performances, quant au temps de reconfiguration, ne sont pas divulguées et aucun exemple d'application n'a

été démontré. Il devient donc difficile de comparer ce système avec ceux des autres recherches. L'année suivante, Bittner et Athanas publient un nouvel article sur leur puce Colt, mais encore une fois, les auteurs n'ont pas illustré les aspects de reconfiguration dynamique de leur système, ils se sont plutôt concentrés sur la présentation de l'architecture du composant.

Dans le même ordre d'idées, une équipe de Northwestern University, en Illinois (Hauck et al., 1997) a présenté un système du nom de Chimaera. Le système propose une solution au problème d'engorgement qui se présente souvent dans le transfert d'information entre un processeur et un FPGA qui joue le rôle de son co-processeur. En combinant l'aspect reprogrammable du FPGA au système hôte, les auteurs croient pouvoir créer un système plus de deux fois plus rapide que les systèmes conventionnels de traitement. Ils veulent, de plus, intégrer au système la propriété de se reconfigurer partiellement pour réduire le temps de reconfiguration de la partie reconfigurable. À partir d'une banque d'instructions complète, le système retient seulement celles nécessaires pour l'opération courante et sert de ce fait de cache d'opération.

Du coté industriel, une équipe de recherche de chez Xilinx (Trimberger et al., 1997) a publié un article à propos d'une nouvelle architecture de FPGA basée sur ceux de la famille 4000 de la même compagnie. Huit configurations pourraient être logées à même ce nouveau FPGA. Ces configurations seraient disposées tout autour du FPGA, ce qui permettrait à celui-ci de se reconfigurer en aussi peu qu'un cycle d'horloge; 30 ns. Les modifications apparentes quant aux fonctionnalités existantes chez le XC4000E sont l'ajout de la possibilité de sauvegarder les états de cellules et une augmentation de la quantité de routage. Les configurations peuvent même être installées lorsque le FPGA est en traitement. Pour contenir ces huit configurations sur le composant et posséder un temps de reconfiguration aussi bas que 30 ns, l'équipe a pourvu chaque cellule de huit blocs de mémoire. Trois modes de configuration sont décrits, soit le mode de temps partagé, le mode "machine logique" (*logic engine*) et le mode statique. Pour le premier mode, un signal interne ou externe permet de déclencher le processus de reconfiguration. Dans le deuxième mode, le processus de reconfiguration est prédéfini et de manière cyclique les reconfigurations se succèdent.

Finalement, dans le mode statique, le contenu des CLB ne change pas. Ce FPGA n'a jamais été commercialisé même si des prototypes ont été construits.

Une équipe de l'Université de Californie à Santa Barbara (Chang et Marek-Sadowska, 1998) a présenté une étude portant sur une nouvelle architecture de FPGA dynamiquement reconfigurable. En particulier, l'équipe s'est penchée sur le partitionnement du système en morceaux de circuits séquentiels, en développant un algorithme de partitionnement. Cet algorithme permet de trouver le partitionnement optimal, qui possède les coûts minimum en matériel et en communication, tout en gardant la performance. Le but de cet article est de supporter l'étude faite sur le développement de l'algorithme, sans toutefois aller vers l'implantation du système. Leur approche de développement est de retourner au niveau porte logique, ce qui permettrait selon eux de construire un réseau de communications processeur/FPGA beaucoup moins coûteux, puisque plus direct. Les performances estimées de leur système montrent qu'ils peuvent réduire le coût relié à la communication de près de 35%.

Un dernier travail de ce type est recensé, il s'agit de celui de Chiricescu et Vai, 1998, du Northeastern University à Boston au Massachusetts. L'article traite du développement d'un FPGA tridimensionnel possédant de la mémoire intégrée pour conserver les données requises pour reconfigurer dynamiquement le système. D'une configuration à l'autre, le système permet de conserver les parties constantes. Le système n'étant pas conçu, il est trop tôt pour connaître les performances réelles de ce circuit.

1.5 ÉTUDES ET ANALYSES SUR L'EFFICACITÉ DE LA RECONFIGURATION DYNAMIQUE

Travailler sur un sujet jeune comme la reconfiguration dynamique peut parfois devenir aveuglant. En effet, comme c'est souvent le cas dans le milieu de la recherche, lorsqu'un sujet devient populaire, une masse de gens se rue vers un travail traitant dudit sujet. Il est bien important de garder à l'esprit que ce n'est pas toutes les

applications qui peuvent tirer bénéfice de l'utilisation de techniques de reconfiguration dynamique. Le système doit pouvoir être séparé de manière séquentielle afin qu'il puisse être exécuté par partie. De plus, certaines applications ont déjà fait l'objet de création de composants dédiés, ce qui pousse loin derrière les performances que peut fournir un système construit sur un FPGA par exemple. C'est pourquoi on retrouve dans la littérature scientifique quelques articles faisant état d'une analyse ou une réflexion sur les caractéristiques d'un système avantagé par l'utilisation d'une technique de reconfiguration dynamique. Il sera intéressant de remarquer la qualité de ces études par rapport à l'époque à laquelle celles-ci sont publiées. On y verra qu'une certaine maturité émane des études plus récentes, mais il reste que le sujet est peut-être encore trop jeune et l'engouement trop fort pour bien percevoir les avantages réels de la reconfiguration dynamique.

Une toute première étude qui analyse les conséquences de la reconfiguration dynamique est publiée par Lysaght et Dunlop de l'Université de Strathclyde à Glasgow en Écosse (1993). Les auteurs précisent d'abord les termes "reconfigurable partiellement" et "reconfigurable dynamiquement". Une figure intéressante est la figure 1.1, publiée avec le même article.

Les auteurs considèrent, selon ce schéma, que la seule manière efficace de faire de la reconfiguration dynamique d'un système est l'utilisation de FPGA partiellement

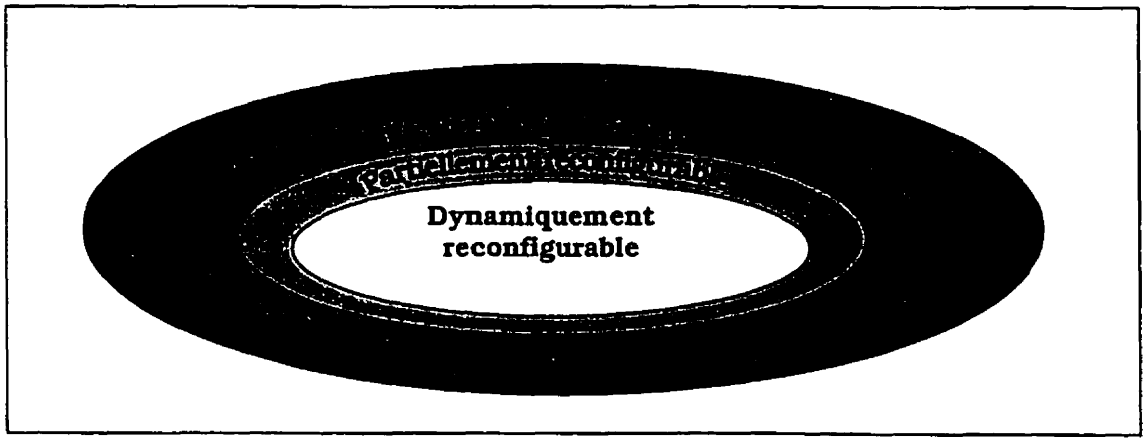

Figure 1.1 Classification des FPGA selon leur configurabilité

reconfigurables. Cependant, comme nous l'avons relaté dans la section 1.2, Lysaght a publié (1991) un article portant sur l'utilisation de FPGA conventionnels comme base de système dynamiquement reconfigurable. C'est là qu'il faut faire une distinction entre FPGA reconfigurable dynamiquement et système reconfigurable dynamiquement. Les auteurs relèvent cependant un point intéressant dans les avantages de la reconfiguration partielle: ils n'écartent pas complètement l'utilisation de FPGA conventionnels pour construire des systèmes reconfigurables dynamiquement, par contre, ils mentionnent que plus la complexité et la taille des FPGA sont imposantes, plus le temps de reconfiguration devient long. C'est pourquoi ils considèrent que la seule manière de construire des systèmes reconfigurables dynamiquement lorsque la taille des FPGA sera encore plus importante devrait être l'utilisation de la reconfiguration partielle. De la même manière, on pourrait considérer que les FPGA de la famille XC6200 de Xilinx sont trop petits pour considérer une implantation réaliste d'un système reconfigurable dynamiquement. En effet, parmi tous les articles cités, aucun ne fait de réelle implantation d'application, complexe et volumineuse sur ces FPGA. Rappelons que les systèmes de taille comme le projet du génome humain de Lemoine et Merceron en 1995 ou encore le système GANGLION de Cox et Blanz, 1992 utilisent des FPGA conventionnels pour les réalisations concrètes. L'énergie se concentre alors uniquement sur le développement du contrôleur de configuration, puisque les outils de placement/routage sont efficaces pour ce type de FPGA, tandis que les utilisateurs de FPGA reconfigurables partiellement doivent investir du temps dans le développement et la réalisation d'un placement/routage serré et complexe, puisqu'il doit imbriquer plusieurs configurations et conserver le plus de logique fixe possible entre chacune d'elles.

Finalement, Lysaght et Dunlop font un bilan des manques de la technologie actuelle (1993) qui limiteraient une réalisation rapide de système efficace et performant, utilisant la reconfiguration dynamique. D'abord, les auteurs mentionnent qu'aucun langage de spécification ne permettrait à ce jour de considérer la dimension qu'apporte la reconfiguration dynamique. C'est toujours le cas en 1999. Ils mentionnent également que les FPGA actuels ne sont pas conçus pour être reconfigurés rapidement et qu'ils sont petits. Ils soulèvent qu'aucun mécanisme de reconfiguration ne peut être déclenché de manière interne au FPGA. En effet, si l'on veut pouvoir construire un système plus performant en terme de reconfiguration

dynamique, il serait intéressant que celui-ci soit en mesure d'effectuer sa propre reconfiguration, une autonomie que distinguerait ces systèmes de tous ceux qui sont disponibles aujourd'hui. Plusieurs autres points sont soulevés par les auteurs, mais, pour les résumer, mentionnons seulement qu'un manque d'outil logiciel considérant l'aspect de la reconfiguration dynamique, autant au point de vue simulation que développement ou placement/routage automatique est soulevé.

Un groupe de l'Université Brigham Young de l'Utah (Withlin et Hutchings, 1997 et 1998) s'est particulièrement penché sur le rendement que peut fournir l'implantation d'une fonctionnalité de reconfiguration dynamique sur quelques types de systèmes. Ils ont développé un indice de densité fonctionnelle (*functional density*) qui leur permet de quantifier les bénéfices reliés à la reconfiguration dynamique. Leur indice de densité fonctionnelle est défini comme suit:

$$D = \frac{1}{AT}$$

Où A est l'aire utilisée du circuit et T le temps d'opération. Le temps d'opération T peut être à son tour décortiqué en deux termes, soit t_{exec} et t_{config} pour les temps d'exécution du système et temps de configuration. Pour un circuit qui procède à une série de reconfigurations, il est clair que la variable temps d'opération sera plus élevée que pour un circuit statique, puisque celui-ci ne consomme pas de temps pour effectuer des reconfiguations, la configuration initiale étant omise. Ainsi donc, on pourra tirer avantage à utiliser la reconfiguration dynamique par rapport à un système fixe si celui-ci peut réduire la quantité de silicium utilisée pour un même système. De fait, un avantage à l'utilisation d'une technique de reconfiguration dynamique est l'économie réalisée avec la réutilisation des ressources matérielles, i.e. réduction de la surface requise pour implanter le système. Notons aussi un autre fait important qui est pris en compte dans la mesure de densité fonctionnelle, il s'agit du temps d'exécution. Cet aspect est moins évident à visualiser mais, pour certains systèmes, un bon partitionnement du circuit peut résulter en une réduction du temps d'exécution global du système.

Étant donné que l'évaluation de la mesure de densité fonctionnelle peut être difficilement réalisable pour les deux types de système, soit ceux qui sont reconfigurables dynamiquement et ceux qui sont statiques, les auteurs ont développé dans leur publication de 1998 une méthode pour estimer la densité fonctionnelle des deux types de circuit. Selon les auteurs, cette évaluation peut s'effectuer en estimant d'abord la surface et le temps de reconfiguration du système utilisant seulement la reconfiguration dynamique.

1.6 Discussion et Conclusion

Les recherches présentées dans ce chapitre couvrent l'essentiel de la matière qui a été publiée depuis le début des années 90 sur le sujet. Auparavant, la reconfiguration dynamique restait une théorie à l'état embryonnaire, qui n'attendait que le réveil du milieu industriel pour se développer. Mais qu'en est-il aujourd'hui? La révolution que nous attendions n'est pas venue. Est-ce que le RTR doit rester à tout jamais une curiosité scientifique? Certains auteurs mentionnaient que la reconfiguration dynamique des FPGA palliait à l'inflexibilité des systèmes dédiés comme les ASIC (Application Specific Integrated Circuit), ainsi qu'à la lenteur des microprocesseurs. Ils voyaient de ce fait un avenir très prometteur pour le FPGA qui s'était jusqu'à maintenant trop souvent contenté du rôle de logique de colle. Voilà près de 10 ans maintenant que le concept de reconfiguration dynamique existe et on peut considérer que celui-ci a atteint un certain niveau de maturité. Pourtant, aucun FPGA disponible commercialement et qui fournit la reconfiguration partielle ne s'est démarqué sur le marché. Cette réalité peut être causée par bien des choses. En effet, il peut y avoir un manque d'intérêt au niveau de l'industrie, si la reconfiguration dynamique possède un nombre trop restreint d'applications. Il peut y avoir un manque au niveau des outils de développement pour la conception, la simulation et le placement/routage de systèmes basés sur des FPGA reprogrammables partiellement. Finalement, il peut y avoir un manque au niveau de l'architecture des FPGA offerts.

Pour la première cause, plusieurs articles disponibles dans la littérature nous prouvent que, au contraire, pour certaines applications bien spécifiques, la

reconfiguration dynamique procure un avantage considérable pour le système, que ce soit au niveau de la réduction de surface de silicium utilisée ou pour la réduction de temps d'exécution (optimisation des circuits). À la seconde cause possible du manque d'intérêt manifeste de la reconfiguration dynamique, soit le manque au niveau des outils de développement, la quantité d'articles sur le sujet parle d'elle-même. En effet, les auteurs qui publiaient sur des applications réalisées à partir de FPGA reconfigurables partiellement, dénoncent le manque d'outils. Par la suite, certains se sont uniquement concentrés au développement d'outils de travail de niveau système, sans toutefois arriver à quelque chose de complètement fonctionnel. Il est clair que la tâche n'est pas simple: créer un outil capable de considérer plusieurs sous-circuits imbriqués de manière à former un tout; concevoir un outil de placement/routage capable de conserver la plus grande surface de logique fixe entre chaque configuration, afin de minimiser le temps alloué à la reconfiguration du FPGA. Même les compagnies productrices de tels FPGA n'ont jamais poussé le développement de tels outils. Par contre, au point de vue de l'utilisation des FPGA conventionnels, pour concevoir des systèmes reconfigurables dynamiquement, l'effort ne doit être fourni qu'au niveau du développement du contrôleur de configuration, puisque tous les outils de placement/routage sont complètement rodés et efficace pour ceux-ci. Aucune équipe ayant réalisé un système à base de FPGA conventionnel n'a mentionné un manque au niveau des outils. Bien sûr, un certain développement minimal est requis puisque propre au circuit, mais outre ce fait, le temps de réalisation global de ce type de systèmes ne dépasse certainement pas celui des systèmes conçus à l'aide de FPGA partiellement reconfigurables.

Finalement, comme dernière cause possible, nous avons évoqué la possibilité d'un manque au niveau de l'architecture même des FPGA partiellement reconfigurables. Les FPGA XC6200 de Xilinx sont des FPGA à grains fins, c'est-à-dire qu'ils permettent d'implanter plusieurs fonctions très simples, à base de seulement deux variables comme entrée. Par contre, la taille des FPGA disponibles dans cette famille ne permettait pas d'implanter des circuits très volumineux. La tendance du marché se dirige présentement vers l'opposé des petits FPGA, surtout avec les tout nouveaux Virtex de Xilinx, dans lesquels on peut implanter jusqu'à un million de portes logiques, nous sommes loin des 65k qu'offrait le plus gros des FPGA de la famille XC6200. Remarquons que depuis que des FPGA reconfigurables

dynamiquement sont apparus sur le marché, on ne retrouve plus de travaux de recherches réalisés à partir de FPGA conventionnels. C'est donc un virage unilatéral que la communauté scientifique a pris, mais ce virage ne se dirigeait peut-être pas nécessairement dans la bonne direction.

Donc, pour répondre a la question que nous avions posée plus haut, à savoir pourquoi la reconfiguration dynamique ne fait pas lieu d'engouement de la part des utilisateurs de système à base de FPGA, c'est probablement un peu dû aux deux dernières causes que nous avons mentionnées: des outils non-performants pour les FPGA reconfigurables dynamiquement et des FPGA trop petits.

L'étude que nous proposons dans ce mémoire tente de répondre principalement au manque sur le marché de systèmes supportant la reconfiguration dynamique de FPGA. Pour ce faire, nous proposons un système qui utilise des FPGA conventionnels de la famille XC4000 de Xilinx qui sont, encore à l'heure actuelle, les FPGA les plus utilisés sur le marché. Le système sera élaboré de manière plus détaillée aux chapitres 2 et 3.

CHAPITRE DEUXIÈME - DÉVELOPPEMENT

2.0 INTRODUCTION

Ce chapitre vise à démontrer le concept à la base du système qui a été implanté dans le cadre de ce projet de maîtrise. L'élaboration du système proposé repose sur une certaine quantité de prérequis et cela est en partie dû aux contraintes physiques de la carte dans laquelle celui-ci a été implanté. Nous tenterons de mettre en relief les aspects avantageux de ce système par rapport à ceux auxquels nous avons fait référence dans le chapitre précédent. Notons que ce chapitre couvre le développement et les hypothèses qui étaient posées avant la réalisation du projet.

Le présent chapitre sera subdivisé comme suit: dans un premier temps, nous exposerons le problème tel qu'il nous a été soumis. Nous consacrerons également une partie du chapitre à présenter la carte dans laquelle l'implantation a été effectuée. Nous exposerons ensuite un protocole sur lequel le concept a été basé, soit le lien de communication JTAG (IEEE 1149.1). Nous discuterons brièvement des problèmes que nous anticipions au tout début du projet, ainsi que les résultats que nous comptions obtenir. Étant donné qu'une des contraintes de départ était l'utilisation de FPGA de la famille XC3000 de Xilinx, nous allons également nous pencher sur l'architecture de cette famille de FPGA.

2.1 EXPOSÉ DU PROBLÈME

La compagnie *MiroTech Microsystems Inc.* de Ville Saint-Laurent produit des cartes munies de logique programmable. Une de leurs cartes, la X-C436, est un co-processeur destiné à accélérer des systèmes qui utilisent les processeurs DSP (*Digital Signal Processing*) TMS320C4x de Texas Instruments. Sa physionomie est de type *daughter board* ou mezzanine, c'est-à-dire qu'elle est conçue pour être juxtaposée sur une carte maîtresse possédant ou offrant l'accès à un processeur 'C40. Elle est surtout utilisée pour accélérer des applications reliées au traitement d'images ou de données.

La carte possède deux unités de traitement, nommées VPE1 et VPE2, pour *Virtual Processing Element 1 et 2*. Ces VPE sont en fait des FPGA XC4036EX de Xilinx. La carte possède également deux modules de soutien, le CSU pour *Configuration and Shutdown Unit* et le CPM pour *Communication Port Manager*. Le CSU est le contrôleur de la carte, il procède à la configuration des deux unités de traitement. Les unités de traitement que possède la carte, effectuent le travail demandé par le processeur principal. On peut utiliser les deux unités à la fois et celles-ci peuvent travailler de manière complètement indépendante ou posséder des configurations entrelacées comme un contrôleur dans l'un et un chemin de données dans l'autre. Lors de la mise en marche de la carte, les configurations des unités de traitement doivent être introduites dans celles-ci. Par contre, pour changer ces configurations après le démarrage initial de la carte, il faut procéder à un arrêt de traitement et reconfigurer les VPE. La figure 2.1 représente le schéma bloc de la carte X-C436 avant l'implantation d'une nouvelle fonctionnalité.

MiroTech nous a mandaté pour introduire un nécessaire de reconfiguration dynamique sur la carte en ne modifiant aucunement le PCB (Printed Circuit Board) de celle-ci. Les seuls champs qu'il était possible d'utiliser sont ceux qui contrôlaient la configuration interne des modules CPM et CSU. Ces derniers sont implantés respectivement sur des FPGA XC3090a et XC3195a de Xilinx. Le type de reconfiguration dynamique dont il est question ici est du type total, puisque nous pratiquerons ces reconfigurations sur des FPGA de la famille XC4000, les unités de

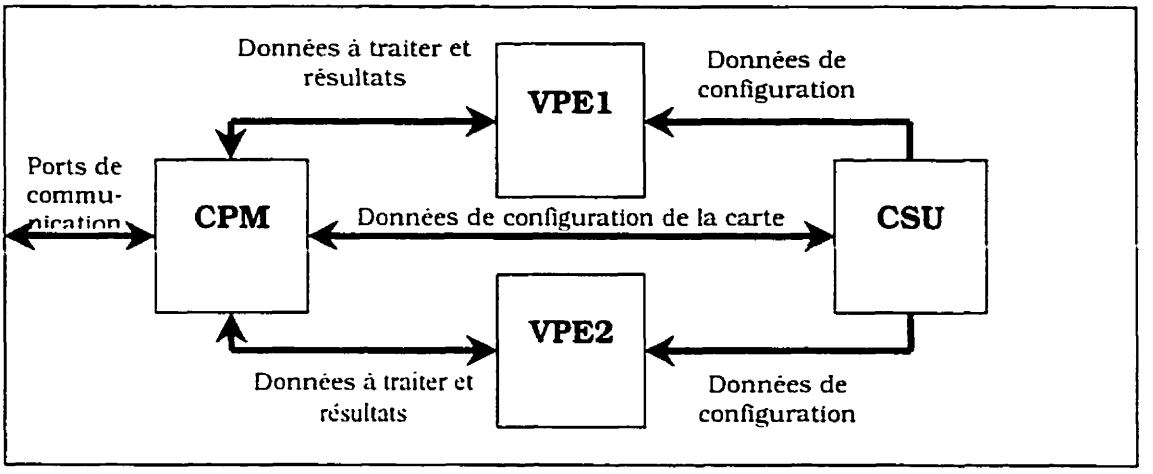

Figure 2.1 Schéma bloc de la carte X-C436

traitement de la carte, et que cette famille de FPGA ne supporte pas la reconfiguration partielle.

Le système repose sur le fait que la carte possède deux unités de traitement. Puisque les unités de traitement peuvent être indépendantes l'une de l'autre, on peut utiliser une des unités pour effectuer un certain traitement, tandis que l'autre est libre pour être reconfigurée. En alternant de la sorte; une unité qui travaille, tandis que l'autre change de configuration, on peut obtenir un système dynamiquement reconfigurable et y implanter des circuits aussi gros que souhaité. À l'instar des systèmes utilisant des FPGA reconfigurables partiellement, notre système ne nécessite aucun temps mort d'un point de vue externe pour procéder au changement de configuration. Il est important de mentionner que si l'application possède un temps de traitement inférieur au temps de reconfiguration, des temps morts peuvent donc apparaître entre chaque configuration. De plus, il y a plusieurs avantages d'utiliser des FPGA de la famille XC4000 qui sont les plus populaires sur le marché présentement.

Selon l'architecture initiale de la carte, il est impossible de reconfigurer une des unités de traitement en laissant l'autre recevoir des données à traiter. Le projet qui repose sur cette implantation peut sembler simple à première vue, mais les multiples contraintes qui y sont rattachées compliquent considérablement la tâche. De plus, outre la modification du système au niveau matériel, nous avons également travaillé sur l'implantation au niveau logiciel en fournissant à la carte un support adéquat, qui

permet d'effectuer les reconfigurations de manière transparente à l'usager. La sous-section suivante introduit plus en détail le fonctionnement de la carte sur laquelle nous avons implanté le système.

2.1.1 DESCRIPTION DU FONCTIONNEMENT DE LA CARTE AVANT L'IMPLANTATION DE LA RECONFIGURATION DYNAMIQUE

La carte X-C436 utilise uniquement le module CPM pour interfacer avec le processeur maître, habituellement un DSP TMS320C4x. Le protocole des ports de communication qu'utilise cette famille de DSP fonctionne par paquets de 32 bits, mais il transfère des mots de seulement 8 bits à la fois. Le processus de transfert est asynchrone et est plutôt basé sur un système de type demande/réponse entre deux composants.

Le CPM possède deux modes de transfert de données. Le premier mode est actif lorsque la carte est en configuration. Les données sont directement acheminées au module CSU qui, selon le code de préambule du mot, les identifie à une commande servant à configurer la carte. Les commandes disponibles servent à configurer l'horloge de traitement, les unités de traitement en elles-mêmes et la configuration des liens internes du CPM qui, une fois en mode opérationnel, transfère des données aux deux unités de traitement. Une fois que la carte ainsi que les unités de traitement sont configurées, un dernier mot de commande parvenant au CSU lui indique qu'il doit placer maintenant la carte en mode opérationnel. Dans ce mode, les données qui parviennent au CPM sont directement acheminées aux unités de traitement et les données traitées retournent, par le chemin inverse, au processeur principal.

Dans le mode configuration, une des commandes envoyées au CSU concernait la configuration des liens internes que prendra le CPM pour diriger les informations aux unités de traitement. En effet, même si la carte fixe la configuration des FIFO (*First In First Out:* une file) des unités de traitement au CPM, celui-ci est en mesure de modifier sur demande, à la configuration, les liens entre les ports de communication et les FIFO. La figure 2.2 montre en pointillé, à l'intérieur du CPM, des interconnexions

possibles entre les ports de communication et les FIFO. Ces interconnexions sont flexibles, c'est-à-dire qu'elles permettent de combiner n'importe quel port de communication à n'importe quel FIFO. Les ports de communication sont bidirectionnels, bien qu'ils ne transmettent les données que dans une seule direction à

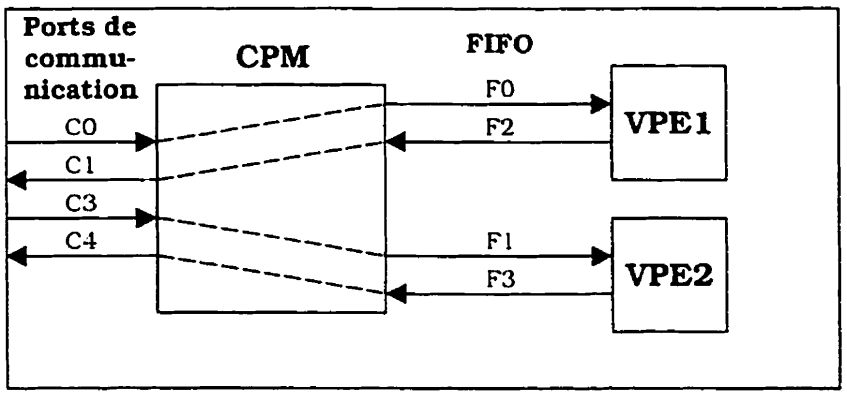

Figure 2.2 Schéma des interconnexions à l'intérieur du CPM

la fois (half duplex), ils sont en mesure de changer de direction lorsqu'une demande de changement de direction est faite. Par contre, les FIFO reliés aux unités de traitement ont été fixés à une seule direction.

Outre les principaux composants, la carte possède également de la mémoire RAM qui est à la disposition des unités de traitement, ainsi qu'une certaine quantité de mémoire DRAM, accessible seulement par le CSU.

Selon l'architecture de la carte avant notre implantation, on pouvait reconfigurer les unités de traitement de la carte, mais cela nécessitait un arrêt dans le traitement de données. En effet, le seul lien de communication avec l'externe exploité était les ports de communication qui, une fois que la carte était en mode opérationnel, transigeait exclusivement avec le CPM. Or, pour effectuer une reconfiguration, nous devons atteindre le contrôleur de la carte, le CSU. Le CPM possède un code qui peut lui indiquer qu'un changement de configuration de la carte doit être effectué. Ce code d'arrêt de traitement est simple: une fois que la carte est en mode opérationnel, les ports de communication deviennent jumelés aux FIFO des unités de traitement. Ces

ports de communication transigent de manière unidirectionnelle, puisque les FIFO fonctionnent de la sorte. Mais étant donné que les ports de communication sont en réalité bidirectionnels, lorsqu'un mot est écrit sur un port de communication configuré en sortie, le CPM, au lieu d'accepter ce changement de direction, comprend plutôt cette action comme une demande d'arrêt de traitement pour la carte. Le CPM communique d'abord un signal aux unités de traitement qui doivent ensuite le retransmettre au CSU. Celui-ci procède ensuite à une remise à zéro de la configuration des deux unités de traitement à la fois.

On voit donc que pour atteindre le CSU et effectuer un changement de configuration des unités de traitement, on doit nécessairement interrompre le transfert de données pour que les commandes qu'envoie le processeur principal arrivent au CSU. Nous devons donc trouver un moyen d'atteindre le CSU sans couper le transfert de données, pour qu'une des unités de traitement puisse continuer à travailler.

Un système supportant la reconfiguration dynamique peut être construit avec les cartes X-C436 dans leur forme originale. Par contre, cela nécessite deux cartes X-C436 au minimum, ainsi qu'un contrôleur externe en mesure de gérer la reconfiguration d'une carte entière pendant que l'autre fonctionne. Par rapport au système que nous voulons implanter, celui-ci nécessite bien entendu le double du matériel, mais cela limite également les types d'applications possibles. En effet, puisque la carte possède une certaine quantité de mémoire pour les unités de traitement et que ceux-ci peuvent l'utiliser en partie ou en totalité, on peut facilement imaginer l'implantation d'un système à base de reconfiguration dynamique, où les données intermédiaires entre chaque traitement sont placées dans cette mémoire commune aux deux unités de traitement. L'implantation d'un tel système ne serait pas possible avec l'utilisation de deux cartes comme base d'un système dynamiquement reconfigurable, à moins de pourvoir ce système d'une mémoire externe aux deux cartes et accessibles par celles-ci.

De plus, mentionnons que l'aspect économie de matériel n'est pas seulement mis en relief pour le gain monétaire d'utiliser une plutôt que deux cartes, mais aussi parce que l'implantation d'un système possédant plusieurs cartes nécessite une plate-

forme suffisamment grande pour être en mesure de faire communiquer le DSP et toutes les cartes du système.

Pour effectuer une reconfiguration dynamique du système, il faut donc être en mesure de conserver une des deux unités de traitement fonctionnelle pendant que l'autre est reconfigurée. Il est clair que l'implantation d'une telle fonction n'est pas possible avec une carte comme celle-ci, puisque pour atteindre le CSU et pour qu'il soit en mesure d'effectuer la reconfiguration d'une des unités de traitement, il faut absolument couper le lien de communication principal, ce qui empêche la suite du traitement. Donc, pour pallier à cette impossibilité d'interrompre le transfert de données pour effectuer une reconfiguration, le système doit être doté d'un deuxième lien de communication. Celui-ci, en plus d'initier une reconfiguration, pourrait faciliter l'échange d'informations pertinentes sur le système traitant. Le type de système que nous voulions créer est représenté à la figure 2.3.

Le lien de communication secondaire que nous voulions utiliser est le lien JTAG. D'abord parce que la carte était compatible à ce protocole jusqu'à un certain point, ensuite parce que c'est un système très répandu, surtout chez les utilisateurs de produits Texas Instruments. La section suivante développe plus en détail le fonctionnement du protocole JTAG.

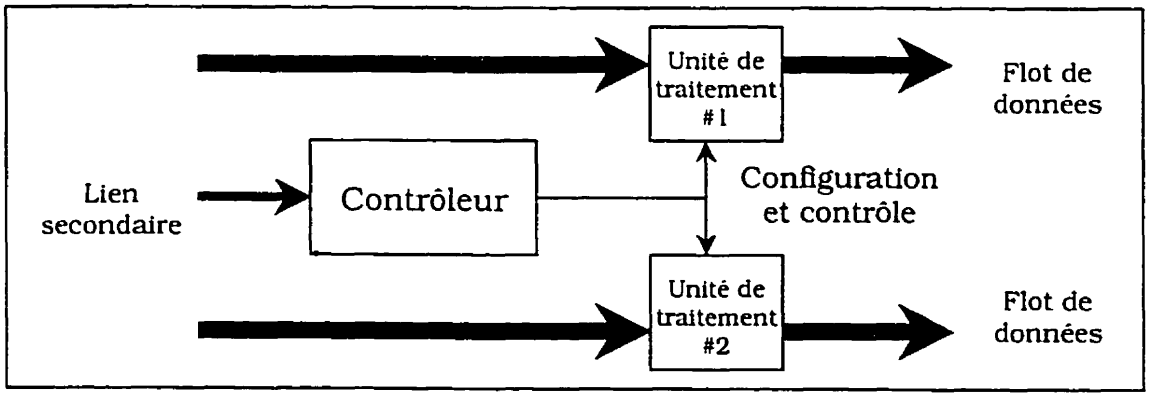

Figure 2.3 Schéma représentant le flot de données principal versus le lien secondaire

2.2 Le lien JTAG, Protocole IEEE 1149.1 Boundary Scan

La première version de ce protocole a vu le jour en 1990. JTAG pour Join Test Action Group est un protocole de plus en plus utilisé, en particulier avec des systèmes utilisant les DSP de Texas Instruments. Le protocole utilise cinq signaux qui relient les différents composants d'une carte ou d'un système de manière à former une chaîne de transmission d'information. La figure 2.4 illustre le concept de base du lien JTAG. L'avantage d'un tel système est que peu importe le nombre de cartes ou de composants qui font partie de la chaîne, aucun changement matériel n'a à être apporté pour satisfaire les règles du protocole. Par contre, quelques petits changements mineurs au niveau logiciel sont requis pour que le transfert de données soit correct. Par contre, le désavantage d'un système de transfert de données sériel est que le délai associé à la propagation des données dépend du nombre de composants dans la chaîne JTAG. Plus le nombre de composants est élevé, plus le délai est long. Les délais engendrés par l'ajout de composants dans la chaîne ne sont pas suffisamment longs pour entraver les performances d'un système qui propage beaucoup de données à la fois et qui utilise une horloge relativement rapide.

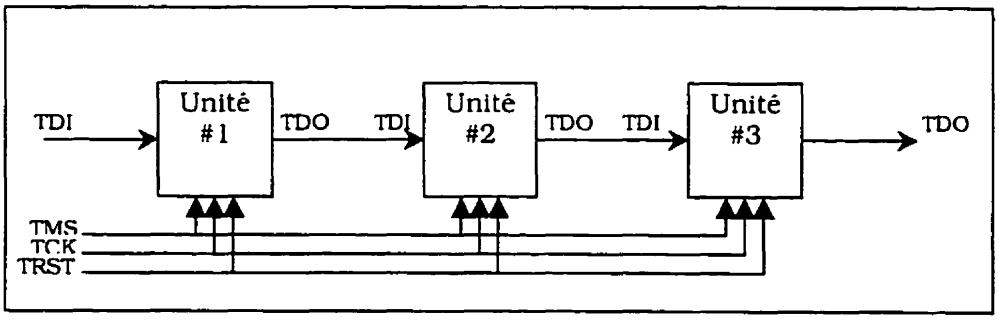

Figure 2.4 Schéma représentant une chaîne JTAG conventionnelle

Le protocole utilise donc cinq signaux, soit quatre entrées et une sortie. Les données, transmises sériellement, parviennent au système avec le lien TDI (*Test Data In*) et ressortent de celui-ci avec TDO (*Test Data Out*). Par la suite, selon la chaîne conçue à l'intérieur de la carte, le signal TDO est transmis à un second composant de la carte sur son lien TDI et ainsi de suite. De manière parallèle, le système possède,

pour tous les composants chaînés, un signal TCK (*Test Clock*) qui est l'horloge du système, un signal TMS (*Test Mode Select*), qui indique de quelle manière ré-assembler les données ou les instructions selon le cas. Finalement, le système possède un signal nommé TRST (*Test Reset*), qui joue le rôle d'une remise à zéro logicielle et dont l'implantation est optionnelle. L'horloge fonctionne typiquement autour de 10 MHz, quelque fois plus lentement, rarement plus rapidement. En effet, pour réussir à propager toutes les données du système, peu importe le nombre d'éléments dans la chaîne, il faut conserver une horloge raisonnable qui évitera les grands biais de synchronisation si le système est plus chargé (i.e., plus de composants dans la chaîne). La machine à états qui représente le processus de reconstruction des données est représentée à la figure 2.5.

Le *TAP Controller* (TAP pour *Test Access Port*) est la machine à états qui doit être implantée dans chacun des composants supportant le protocole JTAG. La transition d'un état à l'autre dépend du signal TMS à la montée de l'horloge et dont on voit la valeur sur chacune des flèches de transition d'états. Deux chemins principaux sont possibles. Le premier, vers la droite, permet d'assembler un registre d'instruction, *IR (Instruction Register)*, tandis que le second permet d'assembler un registre de données, *DR (Data Register)*. Ces registres sont la base même du système JTAG. Donc, une instruction permet de recueillir des données à l'aide du registre de données, selon un format spécifique à l'instruction, mais elle peut aussi permettre de retourner des données selon un format préétabli pour l'instruction. Plus simplement, l'instruction peut déclencher une action dans le composant concerné sans transfert de données. La taille du registre d'instruction est fixe pour chaque composant, tandis que la taille du registre de données varie selon l'instruction courante. Il est important toutefois de noter qu'un composant faisant partie d'une chaîne JTAG ne peut initier un transfert de données. Pour tous les éléments de la chaîne, un contrôleur est assigné au transfert de données et c'est le seul à pouvoir initier un transfert de données. On peut tout de même recevoir des informations spécifiques à un composant, mais il faut dans ce cas que la machine à états à l'intérieur d'un composant soit conçue de la sorte. Cela implique que lorsque l'interface JTAG du composant se trouve dans une instruction particulière, elle peut transmettre sur le lien TDI des signaux variables, dont la nature et l'ordre sont préétablis.

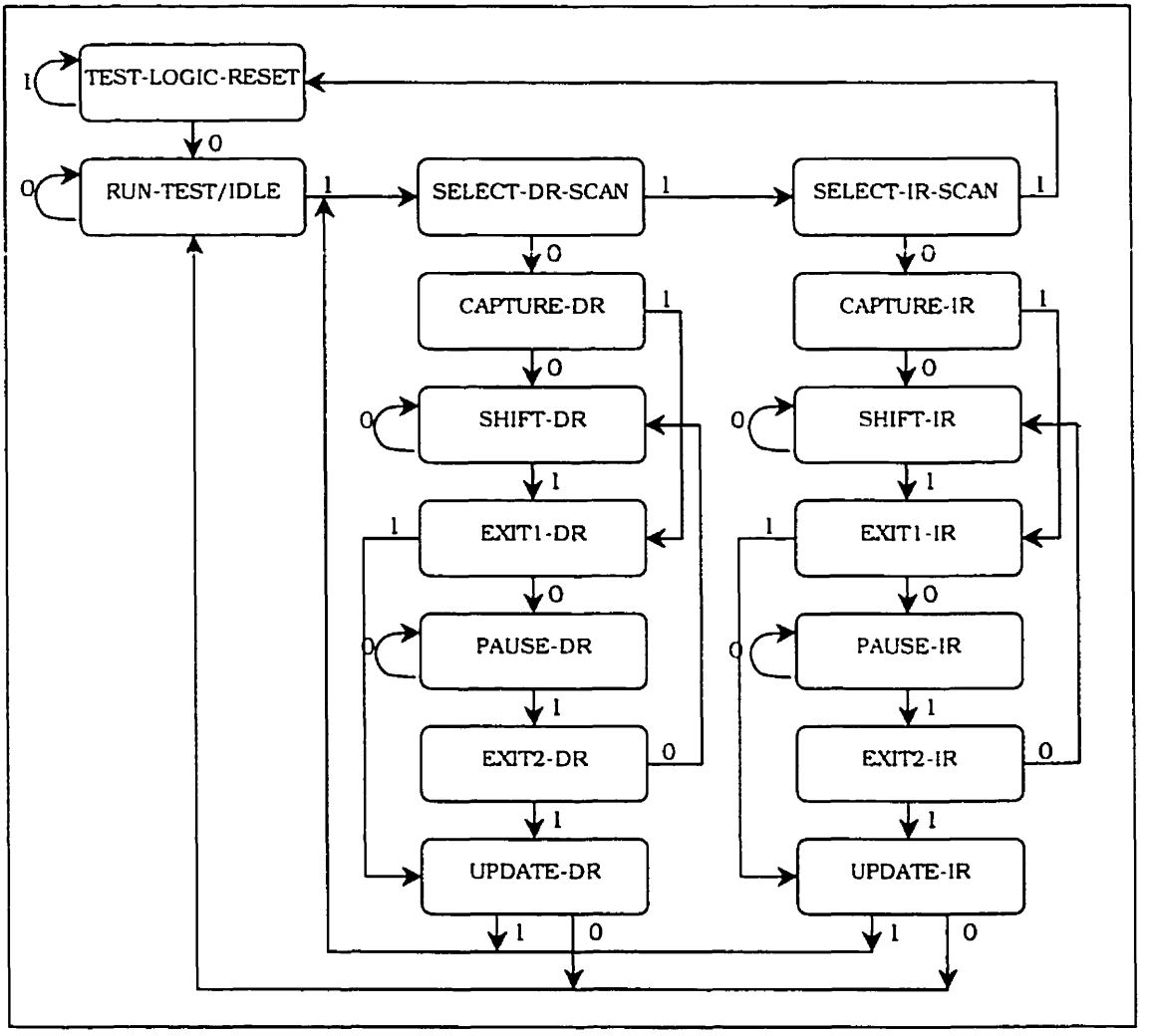

Figure 2.5 Diagramme d'états du *TAP controller.*
Note: Les étiquettes sur les transitions d'états reflètent la valeur du signal TMS.

Si on suit chacune des étapes de la machine à états de la figure 2.5, un registre est d'abord capturé et chargé parallèlement à l'intérieur du registre courant. Un seul cycle s'écoule dans cet état. Durant l'étape *SHIFT*, la suivante, les données sont décalées, selon les bits de poids faibles, et se retrouvent sur la sortie TDO. L'entrée TDI, quant à elle, fournit la nouvelle valeur qui se retrouve sur l'espace libre, le bit le plus significatif. Les premières valeurs à sortir du registre sont celles capturées à l'étape précédente (état *CAPTURE*). La valeur du registre devient fixe et valide à la sortie de l'état *UPDATE* seulement. En effet, les états entre *SHIFT* et *UPDATE* permettent plus de flexibilité à l'utilisateur, mais ils n'altèrent pas le registre. À l'aide des états *EXIT1*,

EXIT2 et *PAUSE*, on ne peut pas effectuer de modification de la valeur du registre. Par contre, on peut retourner à l'état *SHIFT* et ainsi remodeler la valeur du registre choisi. Si on observe en particulier les états *CAPTURE* et *SHIFT*, la valeur de TDO, selon l'entrée TDI, pourrait s'illustrer par la figure 2.6.

Dans cette figure, on peut voir que dans un premier temps la sortie TDO est en haute impédance lorsqu'on se trouve dans l'état *CAPTURE*. Ensuite, la valeur sur TDI se propage aussi longtemps qu'on se trouve dans l'état *SHIFT*. L'état *CAPTURE* permet de renvoyer une information précise sur le système. Par exemple, pour les FPGA de la famille XC4000 de Xilinx, lors du passage de l'état *CAPTURE-IR*, les informations

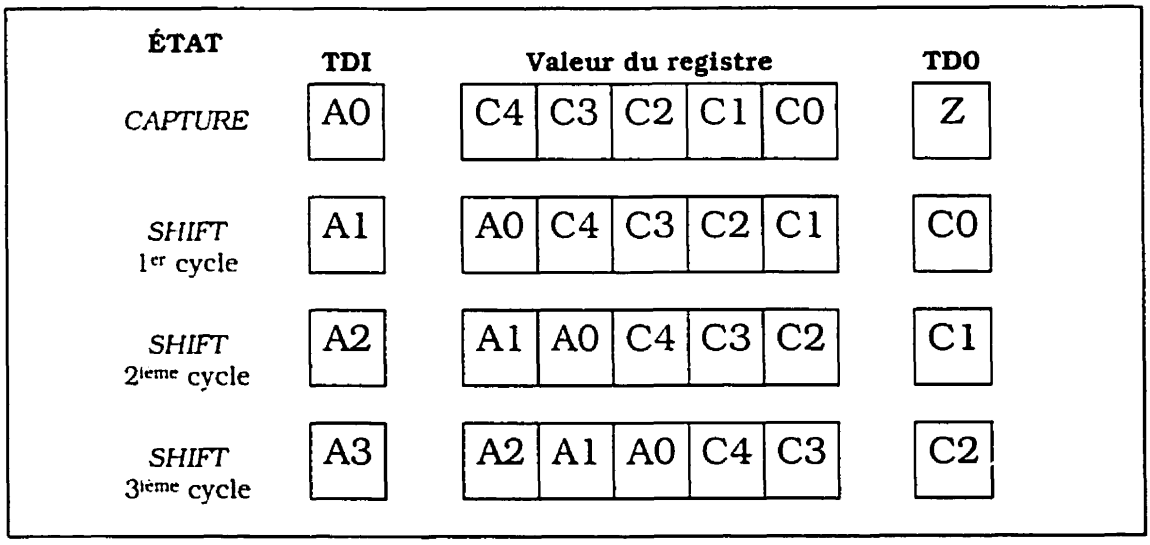

Figure 2.6 Représentation du décalage de données pour les états *CAPTURE* et *SHIFT*

retournées vont concerner l'état actuel de la configuration du FPGA.

Quelques états seulement sont dits "stables", c'est-à-dire qu'on peut demeurer plus d'un cycle dans ces états. De plus, pour ajouter à leur caractère de stabilité, on peut remarquer qu'il n'est pas nécessaire de changer la valeur du signal TMS une fois qu'on entre dans l'état pour y rester. Ces états sont *TEST-LOGIC-RESET*, *RUN-TEST/IDLE*, *CAPTURE-DR*, *CAPTURE-IR*, *SHIFT-DR* et *SHIFT-IR*. Cet aspect est important pour synthétiser un système capable de simuler l'envoi de données selon le protocole JTAG et qui agirait à titre de source des signaux de contrôle et d'horloge. D'ailleurs, un composant disponible sur le marché, le Test Bus Controller (TBC) de

Texas Instruments, permet de transmettre des données selon le protocole JTAG. Pour contrôler ce TBC et lui indiquer quel chemin nous voulons que la machine à états parcoure, nous n'avons qu'à lui préciser, selon certaines commandes, quels états stables parcourir pour transmettre les données. De plus, il permet d'envoyer une certaine quantité de données sans interruption, puisqu'il possède des FIFO internes de longueur variable selon le modèle de TBC. Par contre, pour transmettre un flot important de données, comme un fichier de configuration de FPGA par exemple, le TBC garde la ligne JTAG en *PAUSE*, pour lui permettre de capturer les données à envoyer, de remplir ces FIFO et poursuivre sa requête de transmission.

Des systèmes comme les DSP C40 de Texas Instruments possèdent un registre d'instruction JTAG de 8 bits de large. Par contre, ces instructions sont toutes réservées à l'usage exclusif des développeurs de Texas Instruments et aucun usager ne peut y accéder. De la même manière, certaines familles de FPGA comme les XC4000 et Virtex de Xilinx possèdent une compatibilité au protocole JTAG. Les registres d'instruction de ceux-ci sont respectivement de 3 et 5 bits et sont illustrés aux tableaux 2.1 et 2.2.

Tableau 2.1 Description des instructions JTAG disponibles avec les FPGA de la famille XC4000 de Xilinx

Instruction	Test sélectionné	Description de l'instruction
000	EXTEST	Permet d'injecter des données à l'intérieur du système.
001	SAMPLE	Permet de capturer l'état des entrées/sorties.
010	USER1	À l'usage de l'utilisateur.
011	USER2	À l'usage de l'utilisateur.
100	READBACK	Permet de connaître la valeur du composant Readback préalablement implantée dans le système.
101	CONFIGURE	Permet de configurer le FPGA à l'aide du lien JTAG.
110	Réservé	Réservé.
111	BYPASS	Place le composant en mode BYPASS, ce qui l'exclut de la chaîne pour les tests.

Tableau 2.2 Description des instructions JTAG disponibles chez les FPGA de la famille Virtex de Xilinx

Instruction	Test sélectionné	Description de l'instruction
00000	EXTEST	Permet d'injecter des données à l'intérieur du système.
00001	SAMPLE	Permet de capturer l'état des entrées/sorties.
00010	USER1	À l'usage de l'utilisateur.
00011	USER2	À l'usage de l'utilisateur.
00100	CFG_OUT	Permet d'effectuer un Readback des composants
00101	CFG_IN	Permet d'effectuer une configuration du FPGA
00111	INTEST	Associé à la commande EXTEST
01000	USRCODE	Permet de renvoyer le code d'usager
01001	IDCODE	Permet de renvoyer le code d'identification
01010	HIZ	Permet de placer les sorties en haute impédance lorsqu'en mode BYPASS
01100	JSTART	Envoie la séquence de démarrage d'horloge si StartupClk est TCK
11111	BYPASS	Place le composant en mode BYPASS, ce qui l'exclut de la chaîne pour les tests.
Tous les autres	Réservé	Réservé.

Il est important de distinguer les types de compatibilité au protocole JTAG. Par exemple, la famille de FPGA XC3000 ne possède pas de broches dédiées au protocole mais, étant donné sa nature, l'implantation d'une machine compatible au protocole est une chose simple à réaliser. De plus, lorsqu'on mentionne que les FPGA de la famille XC4000 sont compatibles au protocole, c'est que certaines broches sur le composant sont dédiées à la réception des signaux JTAG et qu'un *TAP Controller* y est déjà implanté. Par contre, la flexibilité que laisse le manufacturier au développeur, à savoir deux instructions *User Defined*, est clairement moins intéressante que d'implanter une interface JTAG propre au système comme nous l'avons fait sur la carte X-C436, mais, elle peut être suffisante pour certain système.

La toute nouvelle famille de FPGA Virtex de Xilinx est elle aussi compatible au protocole JTAG. Ces FPGA possèdent un registre d'instruction de 5 bits de largeur et pourraient, à l'aide du lien JTAG, permettre une reconfiguration par colonne. Pour en revenir aux XC4000 qui sont utilisés comme unité de traitement sur la carte X-C436 de MiroTech Microsystems Inc., nous n'avons pas utilisé le mode *CONFIGURE* des commandes JTAG disponibles pour la réalisation du projet de reconfiguration dynamique. En effet, MiroTech avait déjà expérimenté cette possibilité de reconfiguration par commande JTAG, mais il semblerait que cette fonction ne soit pas encore tout à fait au point et qu'il y aurait une incompatibilité de "*timing*" avec l'emploi d'un TBC pour transmettre les données (Mis en *PAUSE* trop longtemps).

Donc, en utilisant le lien JTAG, nous croyons pouvoir atteindre le contrôleur (CSU) de la carte X-C436, afin de pouvoir lui indiquer qu'il devra effectuer une reconfiguration d'une unité de traitement. On peut également lui envoyer une instruction lui indiquant de retourner l'état de configuration des unités de traitement, puisque si un problème survient durant la configuration, certain signaux peuvent l'indiquer et ils sont accessibles par le contrôleur. En fait, à l'aide de ce lien, on peut construire un système selon nos besoins et il peut aussi contribuer au développement du système. C'est donc avec le protocole JTAG comme base de transfert d'information que nous avons implanté les fonctions nécessaires à la réalisation d'un système dynamiquement reconfigurable.

2.3 FONCTIONNALITÉS DÉSIRÉES ET DIFFICULTÉS PRÉVUES

2.3.1 FONCTIONNALITÉS DÉSIRÉES

À l'aide du lien JTAG, nous avons donc résolu le problème de communication avec le CSU. Il s'agissait par la suite de déterminer la série d'instructions que nous voulions implanter à l'intérieur de celui-ci. Pour effectuer une reconfiguration d'un FPGA, certaines étapes précises sont requises. D'abord, il faut effectuer la remise à zéro du système visé, ensuite, il faut envoyer la configuration en stimulant l'horloge de

configuration et finalement, il faut effectuer une mise en marche du FPGA, ce qui consiste à stimuler le circuit pour quelques cycles l'horloge de configuration. Pour offrir une plus grande flexibilité à l'usager, il serait également adéquat de pouvoir modifier les liens internes au CPM, qui relient les FIFO des unités de traitement aux ports de communication de la carte (figure 2.2). Finalement, un changement dans l'horloge de traitement est parfois nécessaire pour certains systèmes et il serait pratique de pouvoir effectuer cette modification à l'aide du lien JTAG lors d'un changement de configuration.

Au minimum, les commandes implantées doivent permettre d'effectuer la reconfiguration d'une unité de traitement. Par contre, pour ce qui est des autres commandes, leur installation est conditionnelle à l'espace libre à l'intérieur du CSU une fois les commandes de base implantées. Nous avions également mentionné que les commandes implantées peuvent permettre d'interagir avec le ou les systèmes traitants. En effet, la carte possède des liens privilégiés entre les deux unités de traitement nommés "llink". Ces liens comprennent 34 lignes bidirectionnelles qui permettent aux unités de traitement d'échanger de l'information selon l'architecture implantée dans celles-ci. Cependant, ces lignes sont également reliées au CSU qui, jusqu'à maintenant, ne les utilise pas. Devant cette ouverture directe, on peut facilement imaginer un système dont les unités de traitement seraient également en mesure de tirer ou de pousser de l'information pour le contrôleur sur ces lignes et vice et versa. Une interaction en temps réel serait possible avec le système traitant, puisque le lien JTAG pourrait contrôler cet accès pour le CSU.

Par contre, comme nous l'avons déjà mentionné, c'est l'espace disponible une fois les commandes de base implantées qui déterminera la variété d'instructions réalisables. Un élément que nous n'avons pas encore abordé est l'endroit où nous comptons entreposer les prochaines configurations à implanter. Nous avons mentionné en premier lieu que le fichier de configuration serait entièrement transmis par le lien JTAG. Par contre, la carte possède également une certaine quantité de mémoire DRAM, deux méga-octets, et elle pourrait donc contenir environ une vingtaine de configurations d'un FPGA XC4036EX. Cette mémoire ayant un temps d'accès relativement rapide, la vitesse à laquelle on pourrait reconfigurer le FPGA ne dépendrait que de la limite de vitesse imposée par le FPGA. Il y a cependant un aspect

qu'il ne faut pas négliger avec l'implantation de cette solution. En effet, cette mémoire nécessite un certain contrôle, dont entre autres le rafraîchissement de la mémoire qui doit être effectué par le CSU. Le problème d'espace se pose encore ici. Par conséquent, pour un premier essai d'implantation de reconfiguration dynamique, il a été décidé d'exécuter la reconfiguration d'une unité de traitement en envoyant le fichier de configuration par le lien JTAG. Les commandes qui ont donc été implantées dans un premier essai sont au nombre de quatre (l'instruction BYPASS étant une commande de base, donc pas spécifique au système) et elles sont indiquées au tableau 2.3.

Tableau 2.3 Commandes JTAG auxquelles répond le CSU afin d'implanter un système de reconfiguration dynamique

Instruction	Test sélectionné	Description de l'instruction
101	LINK	Reconfigure les liens internes du CPM
100	CFGINFO	Indique quelle unité de traitement sera reconfigurée
011	CFGDATA	Contient les données de configuration
010	STATUT	Informe du statut des signaux de contrôle des deux unités de traitement
111	BYPASS	Place le composant en mode BYPASS, ce qui l'exclut de la chaîne pour les tests.
Tous les autres	Réservé	Réservé

2.3.2 Difficultés prévues

Certaines contraintes étaient déjà imposées au tout début du projet puisque nous implantions le système de reconfiguration dynamique sur une carte existante. D'abord, la physionomie, l'aspect matériel de la carte (PCB), devait rester fixe et seulement le contrôleur (CSU) pouvait être modifié. De plus, la conception du module CPM a été réalisée de manière tellement serrée et ardue que MiroTech ne voulait pas se lancer dans une modification de ce module, qui a pris plus de deux années-personnes

de développement, placement/routage à la main. Il reste sur ce module seulement six CLB (*Configurable Logic Blocks*) libres. Donc, l'intervention doit s'effectuer seulement sur le CSU en y implantant une machine à états compatible au protocole JTAG.

Pour ce qui est du contrôleur, ce système nous forçait à utiliser un FPGA XC3195 de Xilinx, occupé à 60% avec la configuration regroupant les fonctionnalités de base qui devaient également se retrouver dans la nouvelle architecture. Les FPGA de la famille XC3000 ont une architecture dont le grain est légèrement plus fin que les XC4000, mais tout de même plus gros que celui des XC6000. En effet, les CLB de cette famille possèdent deux sorties registrées et cinq entrées. Pour des systèmes tel que le traitement de chemins de données, les FPGA de cette famille peuvent s'avérer adéquats et même fournir d'excellents résultats une fois que le cap du 80 % d'utilisation est dépassé. Par contre, pour des systèmes de contrôle comme le CSU, lorsque le pourcentage d'utilisation devient trop élevé, le routage devient complètement engorgé et rend impossible la réalisation complète de l'étape de placement/routage.

La figure 2.7 représente l'architecture d'un CLB d'un FPGA de la famille XC3000. On peut remarquer que cette famille n'est pas particulièrement spécialisée pour l'implantation d'additionneurs ou de multiplieurs, puisqu'on ne retrouve pas de chaîne de retenue déjà implantée dans le CLB. De plus, les registres à la sortie ne permettent pas d'effectuer de SET au démarrage du FPGA et, pour en utiliser un, il en coûte un peu de logique externe.

La figure 2.8 montre, quant à elle, les possibilités d'utilisation des cinq entrées versus les deux sorties. On voit que pour utiliser les cinq entrées dans une même équation, on doit restreindre le CLB à n'utiliser qu'une sortie sur deux. Pour utiliser les deux sorties, l'équation qui correspond à chacune d'elles doit être composée d'au plus 4 éléments d'entrée.

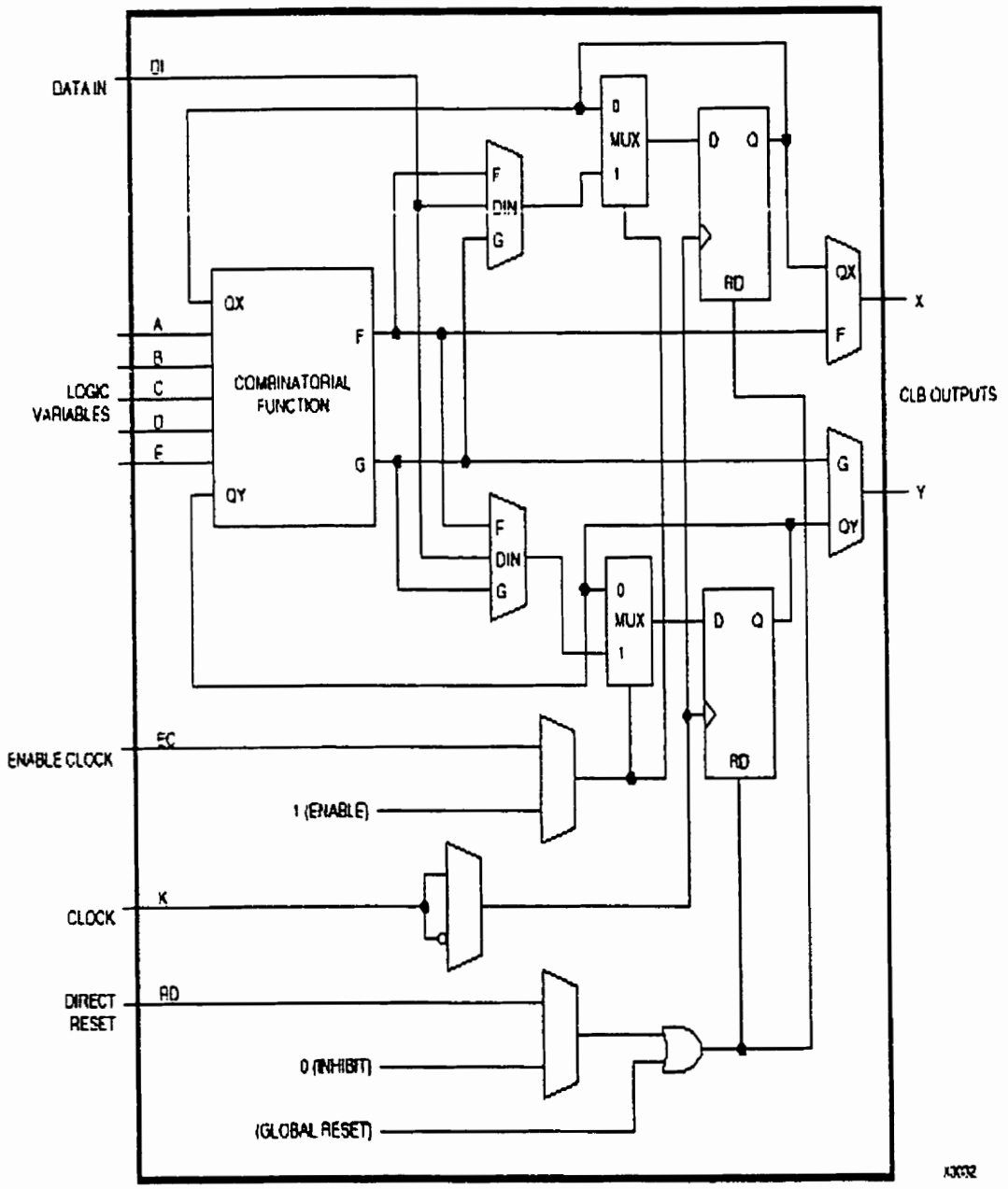

Figure 2.7 Détails d'un CLB pour un FPGA de la famille XC3000 de Xilinx (Tirée de Xilinx Databook, 1998, page 7-9.)

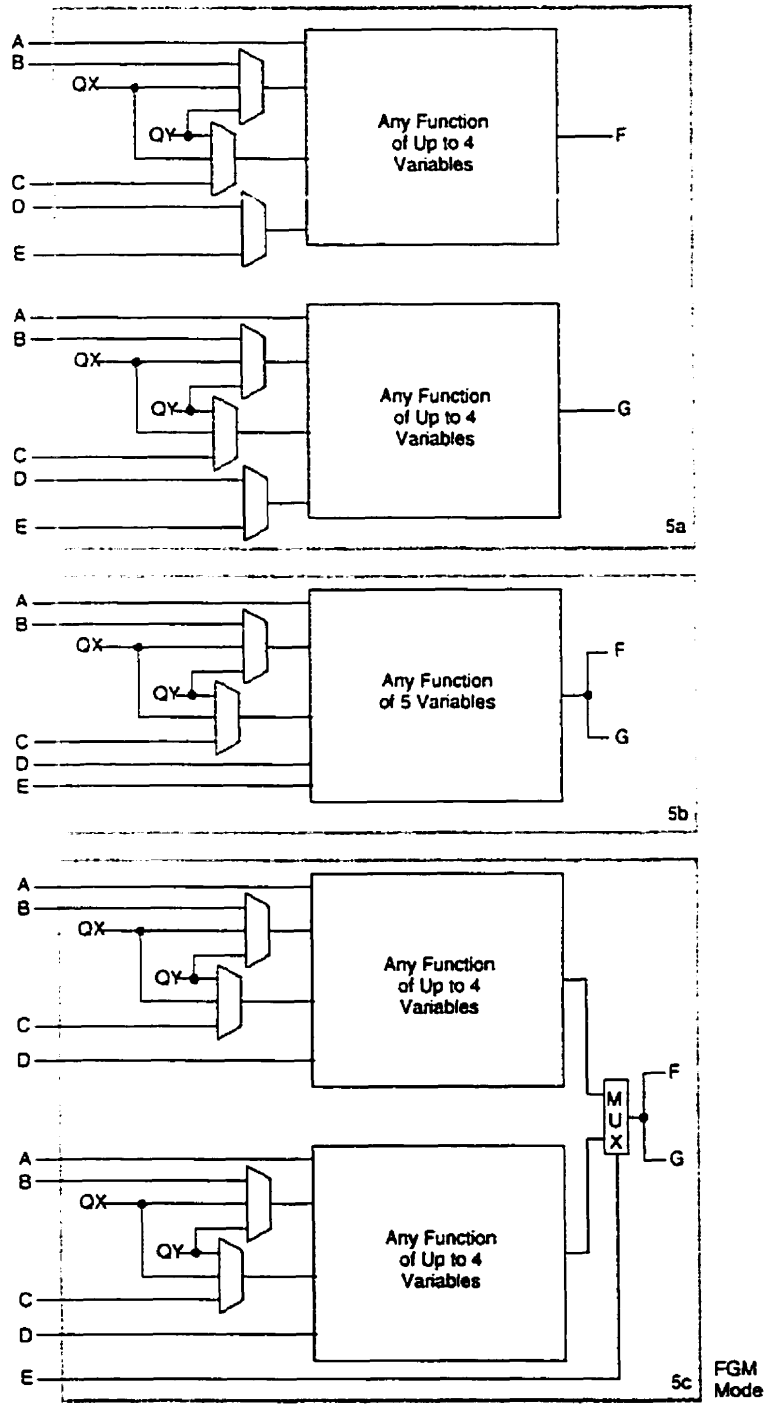

Figure 2.8 Représentation des possibilités de combinaisons pour l'utilisation des cinq entrées et deux sorties d'un CLB de FPGA de la famille XC3000 de Xilinx
(Tirée de Xilinx Databook, 1998, page 7-10.)

Dans l'architecture du FPGA en tant que tel, certaines limitations sont présentes au niveau des composants disponibles, tels les amplificateurs d'horloge. Les FPGA de la famille XC3000 n'en possèdent que deux et ils sont situés l'un à l'extrémité nord-ouest et l'autre au coin sud-est du FPGA. L'architecture du CSU avant l'implantation de notre projet utilisait déjà les deux amplificateurs d'horloge. De plus, cette contrainte se fait ressentir surtout lors du placement/routage et en particulier lorsqu'une bonne majorité du système possède la même horloge, ce qui est le cas avec le CSU. Nous savions que l'introduction d'une nouvelle source d'horloge, telle TCK pour le lien JTAG, exposerait fort probablement une contrainte additionnelle au niveau du placement des composants alimentés par cette horloge, puisqu'aucun amplificateur d'horloge n'est disponible pour TCK et qu'un placement serré sera nécessaire pour limiter les risques d'un important biais de synchronisation.

Le FPGA possède également des ressources de routage assez restreintes, ce qui complique encore plus le placement/routage d'un système qui occupe une grande surface sur le FPGA. Nous avons mentionné que l'architecture du FPGA n'était pas conçue pour supporter facilement des composants tels des additionneurs ou compteurs, puisqu'il n'y a pas de chaîne de retenue. Par contre, le routage externe au CLB est fait pour supporter les structures "à décalage", comme les compteurs et les registres à décalage. De cette manière, les CLB adjacents verticalement ou horizontalement peuvent être facilement liés de sortie à entrée et permettent le décalage des données. Parmi les lignes de routage rapide, notons que les FPGA de la famille XC3000 possèdent trois "*long lines*" par colonne et deux par rangée. Le tableau 2.4 montre les ressources disponibles dans les FPGA de la famille XC4000. On peut constater certaines améliorations dans cette famille, dont l'introduction de lignes verticales dédiées à la propagation du signal d'horloge des CLB, ce qui réserve les « *long lines* » pour le transport de données seulement.

Les figures 2.9 et 2.10 représentent plus visuellement l'architecture de ces différents types de lignes autour d'un CLB. Les lignes simples sont celles qui apportent la flexibilité dans le routage et elles sont les lignes les plus rapides entre deux CLB. Elles ne parcourent que la distance d'un CLB avant de parvenir à une matrice d'interconnexions. Les lignes doubles sont deux fois plus longues que les simples et elles parcourent une distance équivalente à deux CLB avant d'entrer dans la matrice

d'interconnexions. Lignes quadruples qui sont quatre fois plus longues que les simples, possèdent également des amplificateurs à chaque matrice d'interconnexions pour leur permettre de perdre moins de puissance. Finalement, les lignes longues parcourent le FPGA sur toute sa largeur ou sur toute sa longueur sans entrer à l'intérieur de matrices d'interconnexions.

Tableau 2.4 Ressources de routage par CLB chez les FPGA de la famille XC4000

	XC4000E		XC4000X	
	Vertical	Horizontal	Vertical	Horizontal
Simples	8	8	8	8
Doubles	4	4	4	4
Quadruples	0	0	12	12
Lignes longues	6	6	10	6
Connexions directes	0	0	2	2
Horloge globale	4	0	8	0
Chaîne de retenue	2	0	1	0
Total	24	18	45	32

Donc, l'implantation d'un système de type contrôleur dans un FPGA de la famille XC3000 ne laissait pas envisager une tâche facile. De plus, puisque ces FPGA sont tout de même de vieux produits de Xilinx et que les familles XC4000 et Virtex sont beaucoup plus populaires dans l'industrie, le support technique au niveau des outils de développement pour le XC3000 n'est pas aussi peaufiné. Ceci nous a affecté plus particulièrement lors du développement du projet.

En effet, afin de permettre une implantation rapide des nouvelles fonctionnalités reliées à la reconfiguration dynamique du système, MiroTech croyait qu'il était préférable de resynthétiser d'abord le CSU, tel qu'il était avant le projet, à l'aide des nouveaux outils de développement. La version originale du CSU a été

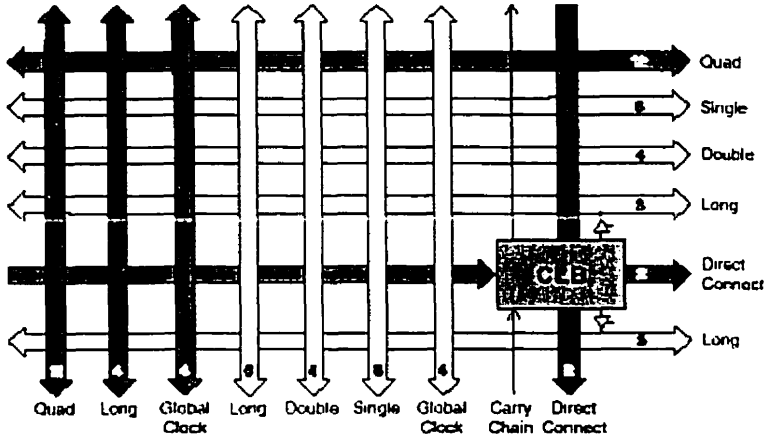

Figure 2.9 Représentation des fils disponibles autour d'un CLB d'un FPGA de la famille XC4000 (Tirée de Xilinx Databook, 1999, page 6-29.)

développée à l'aide des outils XACT de Xilinx, mais cette génération d'outil est morte et a fait place la génération des outils M1. La réimplantation du CSU a permis, entre autres, de connaître toutes les lacunes et particularités des nouveaux outils avec les FPGA de la famille XC3000. Elle a aussi permis de connaître la sensibilité des compilateurs VHDL qui ont, eux aussi, évolués avec les outils. D'ailleurs, certaines lignes de code ont dû être modifiées pour que les compilateurs synthétisent correctement l'objet désiré.

Un bénéfice sûr de la réimplantation du CSU est qu'il nous a permis d'acquérir une plus grande compréhension du système, puisque cette tâche était également accompagnée de la restructuration du banc de test pour celui-ci. À l'intérieur de ce banc de test, tous les composants de la carte y sont présents, incluant le plus important élément du banc de test, le CPM. Afin de pouvoir procéder à des simulations avec délais, nous devions utiliser des outils de Xilinx nous permettant d'obtenir un fichier VHDL à partir du fichier de configuration du FPGA. Ce fichier VHDL est entièrement constitué de composants propres au FPGA utilisé et les délais relatifs à la position des éléments est aussi une information incluse dans le fichier. Cependant, étant donné que nous devions obtenir un fichier de simulation avec délais pour le CPM également et que la conception de celui-ci date de la dernière génération d'outils, nous avons dû utiliser des outils de conversion fournis par Xilinx, nous permettant finalement d'utiliser un fichier VHDL adéquat pour nos simulations. Cette fois-ci, nous n'avons pas remonté le CPM avec les nouveaux outils comme il a été le cas pour le

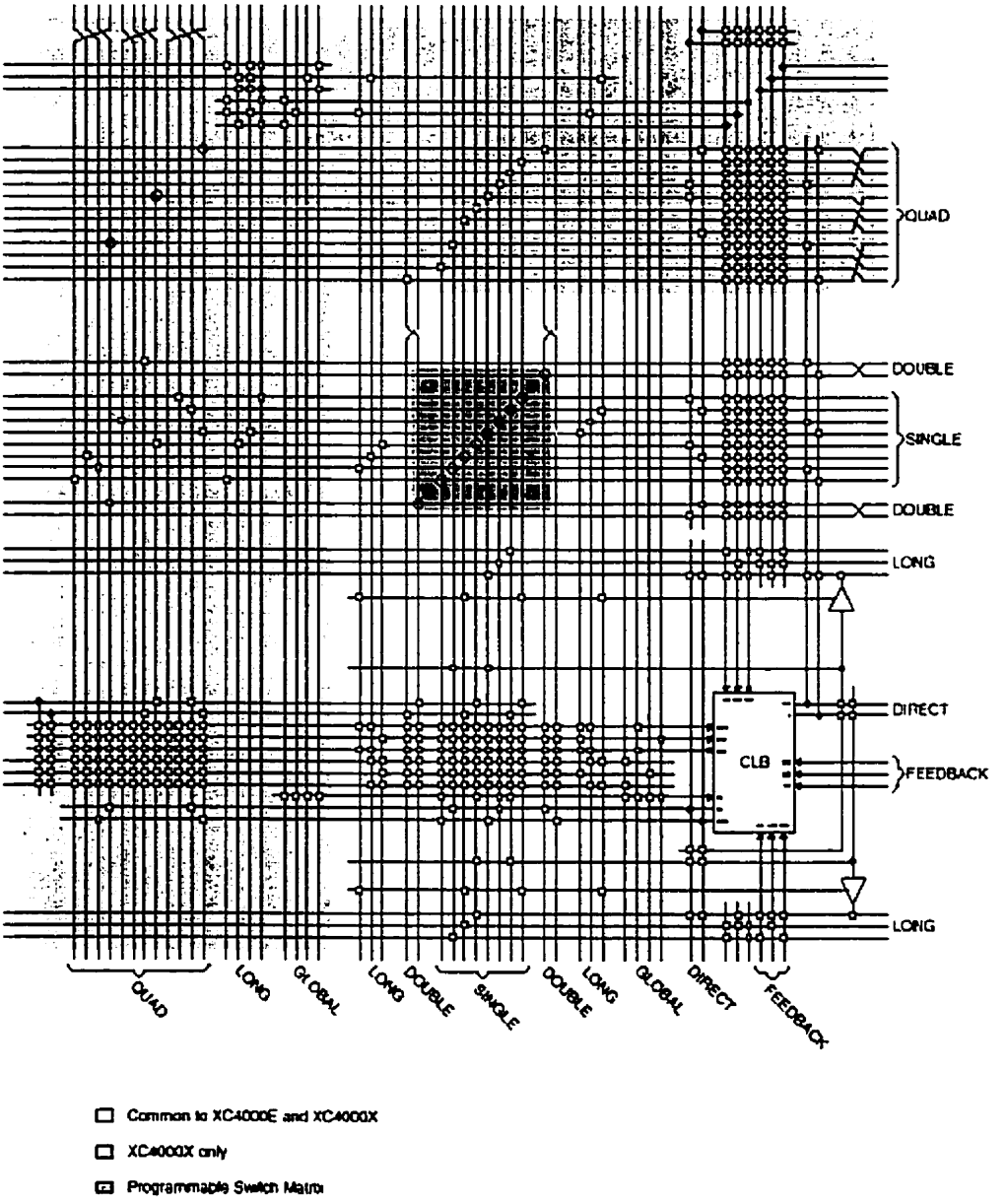

Figure 2.10 Représentation des matrices d'interconnexion ainsi que des différents types de lignes autour d'un CLB d'un FPGA de la famille XC4000.
(Tirée de Xilinx Databook, 1999, page 6-30.)

CSU, nous avons seulement procédé à une conversion de génération. Cette conversion, qui devait en principe être l'histoire de quelques minutes, s'est transformée en un ardu travail de deux semaines. En effet, les outils de conversion ont inséré des erreurs. La détection de celles-ci s'est avérée un travail minutieux, puisque nous avons dû comparer toutes les équations contenues dans tous les CLB. Finalement, nous avons

trouvé que six horloges provenant d'autant de CLB avaient été inversées en polarité durant la conversion et changeaient leur utilisation de front montant à front descendant ou vice versa. Ceci créait un déphasage dans la capture de certaines données critiques.

2.4 Conclusion

À l'intérieur de ce chapitre, nous avons exposé le problème à résoudre en faisant ressortir toutes les contraintes qu'imposaient le système et le manufacturier de la carte X-C436. Puisque nous introduisions un nouveau protocole de communication qui permettait de contourner le lien de communication dédié au processeur principal, nous avons présenté le protocole JTAG; IEEE 1149.1 Boundary Scan, de manière à faciliter la compréhension du lecteur. Afin d'implanter la reconfiguration dynamique sur la carte X-C436, nous avons également exposé les stratégies que nous voulions utiliser en énumérant les commandes qui nous sont nécessaires pour arriver à nos fins. Les aspects présentés dans la dernière section illustrent l'approche réelle que nous avons prise au tout début du projet. C'est dans le chapitre suivant, celui traitant de l'implantation physique du projet, que nous allons aborder tous les détails techniques et les problèmes survenus lors de la réalisation de celui-ci.

CHAPITRE TROISIÈME - IMPLANTATION

3.0 INTRODUCTION

Dans ce chapitre, nous allons traiter de l'implantation proprement dite, sur la carte X-C436, d'un système de reconfiguration dynamique. Nous aborderons l'implantation matérielle et les détails du fonctionnement du CSU, ainsi que l'implantation logicielle et toute la structure qui est impliquée dans la réalisation d'un projet comme celui-ci.

Afin de mesurer les résultats de cette implantation de reconfiguration dynamique, nous avons également travaillé sur une démonstration de traitement d'images. Les détails de cette implantation fournissent plusieurs contraintes au niveau logiciel et matériel et c'est pourquoi nous abordons cette application dans la partie traitant de l'aspect logiciel. Les mesures présentées dans la section résultats seront tirées de cette démonstration. Finalement, nous étudierons les caractéristiques de cette nouvelle structure afin d'en trouver les points faibles et nous proposerons des recommandations en vue d'améliorer le système implanté.

3.1 L'IMPLANTATION MATÉRIELLE

L'implantation matérielle a été une partie critique de ce projet. En effet, l'implantation de la reconfiguration dynamique s'effectuait sur un module existant, le CSU, et nous devions y conserver toutes les fonctionnalités présentes ainsi que l'intégrité de ses performances. C'est donc l'interaction avec les modules existants dans le CSU qui a demandé le plus d'attention. Le CSU est conçu en plusieurs petits modules qui ont une tâche bien spécifique à accomplir et qui interagissent entre eux. Six modules sont présents dans le CSU. Un premier module (PORTSCAN) a comme but de transiger avec le CPM afin de lui faire parvenir certaines instructions ou encore de réassembler les mots que lui fournit le processeur principal, via les ports de communication et le CPM. Un autre module (LICENSE MANAGER) se charge de décoder et de valider la licence de l'usager. Un module (CFG4000) s'occupe de la configuration des unités de traitement. Un autre module (SHUTDOWN) se charge des arrêts de traitement de la carte, tandis que le module "CLOCK GENERATOR" est le générateur d'horloges pour le CPM et les unités de traitement. Finalement, le CSU possède un module (DISPATCHER) qui se charge de décoder les commandes reçues par les ports de communication et de transférer les bonnes informations aux bons modules spécialisés.

La figure 3.1 présente un schéma bloc des modules du CSU et l'interaction entre eux. Aucun nom n'est inscrit sur les signaux internes du CSU pour des raisons de confidentialité qui sont reliées à ce produit. Toutefois, nous pouvons mentionner la nature de certains signaux qui sont dirigés ou proviennent de l'externe. Ainsi, les modules CFG4000s communiquent exclusivement avec les deux unités de traitement (VPE), tandis que le module PORTSCAN interagit avec le CPM. Le module SHUTDOWN reçoit de l'information des unités de traitement et le module JTAG, les quatre signaux nécessaires à la transmission du protocole JTAG soit TDI (*Test Data In*), TDO (*Test Data Out*), TCK (*Test Clock*) et TMS (*Test Mode Select*). La large bande noire reliant plusieurs modules entre eux est le bus de données principal du CSU qui est d'une largeur de 32 bits.

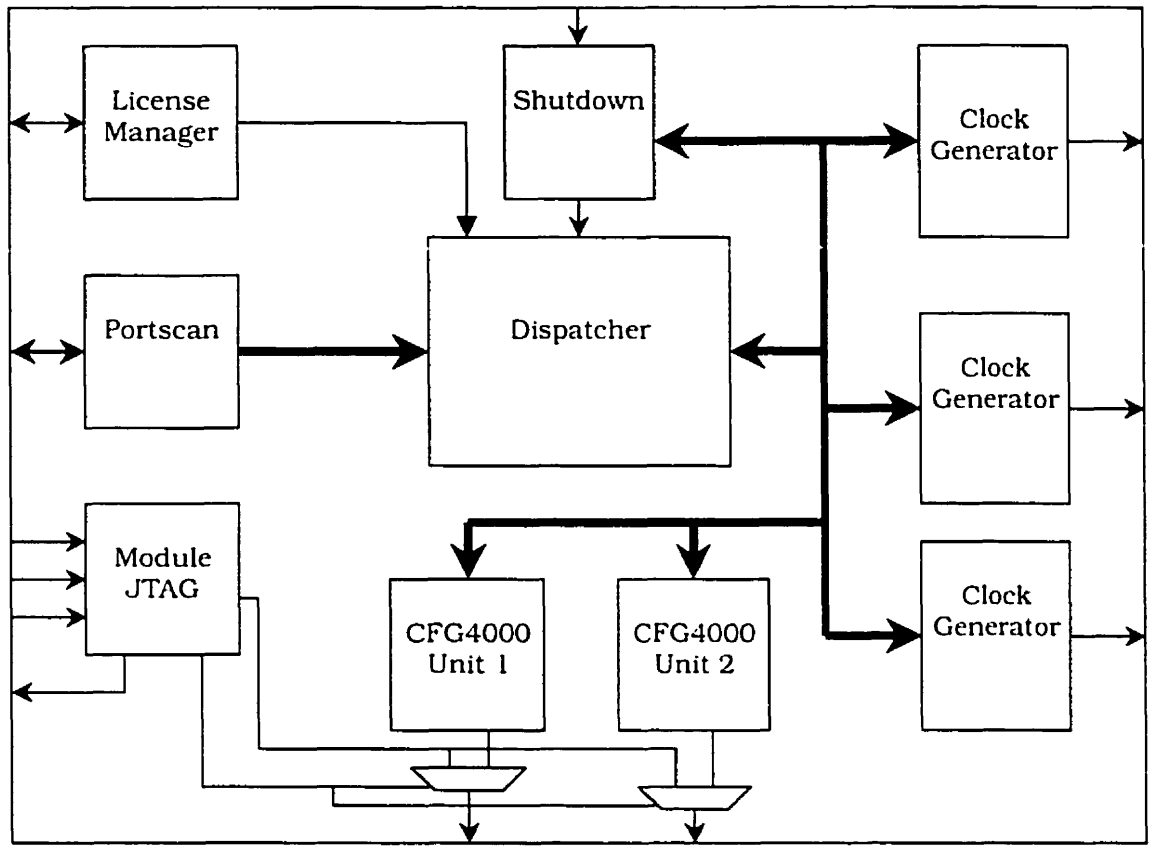

Figure 3.1 Schéma bloc du CSU (*Configuration and Shutdown Unit*)

L'implantation d'un système réalisant la reconfiguration dynamique de manière la plus simple qu'il soit nécessitait une intervention au niveau de la configuration des unités de traitement seulement. Ainsi, nous avions à intervenir au niveau du module de configuration des unités de traitement (CFG4000), en laissant intactes les procédures existantes qui permettent d'effectuer une configuration par les ports de communication. Dans un premier essai, voulant sauver le plus d'espace possible, nous avons réutilisé au maximum la logique en place en intégrant le lien JTAG comme source de données au module existant. En d'autres termes, nous avons essayé de fondre la logique des deux fonctionnalités. Par contre, il s'est avéré que de cette manière, on compliquait encore plus les choses, principalement dû au fait que les deux sources de ce module CFG4000, soit les données des ports de communication et les données du lien JTAG, n'étaient pas sous la même forme (l'une en mot de 32 bits et l'autre sous forme sérielle). De plus, lorsque le module de configuration des unités de traitement (CFG4000) interagit avec les ports de communication, il peut ordonner un arrêt de transfert de données et pour ce faire utiliser, avec le module PORTSCAN, une

forme de protocole de communication basé sur des demandes et des réponses (Request et Acknowledge).

Le fonctionnement est tout à fait différent avec le lien JTAG. En effet, ce lien est commandé exclusivement de l'extérieur et lorsque des données sont envoyées à un composant quelconque, celui-ci doit les traiter ou les ignorer, mais il ne peut en aucun cas les bloquer. La solution à cette partie de l'implantation a été de laisser le module existant tel quel et de placer un multiplexeur à sa sortie dont la sélection est gérée par le lien JTAG. Ainsi, lorsque l'on effectue une reconfiguration de manière conventionnelle, par les ports de communication, le module fonctionne comme avant. Par contre, lorsqu'on utilise le lien JTAG pour reconfigurer une unité de traitement, c'est le lien JTAG qui envoie les données de configuration. Puisque la partie de contrôle matérielle n'est plus possible avec le deuxième lien de communication, les données sont envoyées correctement, selon le protocole de reconfiguration établi, mais de manière logicielle cette fois.

En ajoutant l'interface JTAG où sont implantées les commandes de reconfiguration dynamique de base vues au tableau 2.3, nous portons le pourcentage d'utilisation du FPGA à environ 80%. Les contraintes de routage dans les premiers essais étaient impossibles à rencontrer. La présence de seulement deux amplificateurs d'horloge et le fait que l'un de ces amplificateurs soit occupé par une horloge de 100 MHz est la principale cause à notre problème. En effet, l'outil crée un engorgement au niveau du placement en essayant de minimiser l'espace entre les CLB qui sont reliés entre eux. De cette manière, il bloque les possibilités de routage entre chaque CLB. De plus, selon l'architecture originale du CSU, il y avait une absence totale du moindre pipelinage et cela fait en sorte qu'il existait des chemins trop longs entre certains CLB et que les contraintes temporelles devenaient impossibles à rencontrer. Par contre, pour un FPGA rempli à 60%, les contraintes peuvent être rencontrées avec un peu de placement manuel des structures critiques comme c'était le cas avec l'architecture première du CSU. Par contre, lorsque le pourcentage d'utilisation touche 80 %, comme dans notre cas, même en fournissant un sérieux effort de placement à la main, les contraintes ne sont pas rencontrées.

Certaines structures sont critiques pour le placement. Notons entres autres que puisque le transfert de données avec les ports de communication est sur une base de 32 bits, toute l'architecture du CSU est basée sur cette largeur de données. Donc, on retrouve un registre 32 bits pour le module qui assemble les mots envoyés par les ports de communication (module PORTSCAN). Cette structure a la double fonction de pouvoir être décalée et chargée parallèlement selon l'usage. Ceci nécessite un placement en chaîne selon les liens privilégiés entre les CLB dans le FPGA. De plus, pour chaque module de configuration des unités de traitement (deux modules CFG4000), un registre à décalage de 32 bits de large doit être lui aussi contraint à une structure particulière et un placement manuel s'impose. De plus, les deux registres à décalage, appartenant aux modules de configuration des unités de traitement, tirent leurs données du registre du module PORTSCAN. Ainsi, le placement de ces trois registres doit être fait de manière relativement serrée entre les structures nommées, tout en conservant suffisamment d'espace pour ne pas engorger le routage. Notons qu'une structure tel un registre à décalage de 32 bits utilise 16 CLB (deux bascule-D par CLB), tandis qu'un registre comme celui du module PORTSCAN qui possède d'autres fonctionnalités, utilise 32 CLB.

Il existe trois générateurs d'horloge dans le CSU. Selon l'architecture du CSU, deux horloges sont nécessaires pour alimenter ces trois générateurs, soit celle à 100 MHz, qui est utilisée exclusivement par ces modules, et l'horloge globale du CSU qui est fixée à 8.33 MHz. On peut facilement imaginer qu'avec une horloge aussi rapide que 100 MHz, le placement de ces trois générateurs d'horloge doit être particulièrement serré. Mais ce n'est pas tout, les générateurs sont conçus sur une base d'une chaîne de bascules D, un compteur en anneau. Cet élément est également très sensible au placement pour tout ce qui a trait aux délais entre les différents éléments. Ainsi, ce n'est pas seulement au niveau du générateur que les contraintes doivent être exercées, mais également à l'intérieur de celui-ci.

Finalement, comme nous l'avions mentionné, l'absence d'un amplificateur d'horloge pour le lien JTAG et son horloge TCK font en sorte que nous devons contraindre l'outil à placer les composants qui sont alimentés par cette horloge de manière à prendre le moins de place possible. Ainsi, en utilisant des lignes rapides pour router cette horloge et en restreignant la longueur des fils, on peut être en

mesure de respecter les contraintes liées à l'horloge et de réussir à obtenir une fréquence d'au moins 10 MHz pour celle-ci, sans accumuler un trop grand biais de synchronisation.

On peut facilement imaginer que le travail pour réussir à rendre fonctionnel le CSU n'a pas été une mince affaire. D'abord, nous avons pipeliné tous les échanges de signaux entre les différents modules du CSU pour raccourcir la longueur des chemins à l'intérieur de celui-ci et relaxer de cette manière les contraintes de l'horloge principale. Ce travail a été délicat puisqu'il fallait respecter tous les délais nécessaires et ne pas perturber les échanges de signaux ponctuels dans certains cycles des machines à états touchées. De plus, le CPM possède des délais particuliers quant à sa communication avec le CSU. Ces délais sont très sensibles et lors du routage du CSU ou lors de la modification des délais reliés au pipelinage, nous avons éprouvé quelques difficultés à rencontrer les exigences du CPM, qui lui, ne pouvait être modifié.

Tous ces détails ont fait que l'implantation matérielle a été une tâche moins longue que l'implantation de l'ancien CSU à l'aide des nouveaux outils. Elle a tout de même consistée en un travail de deux mois. Toutes les commandes JTAG que nous voulions implanter lors de ce premier essai ont pu être réalisées. Par contre, un problème survient lorsque nous voulons exécuter la commande de changement des liens internes du CPM, la commande LINK de la série d'instructions JTAG implantée. En effet, ceci constituait un premier essai chez MiroTech de changer les connexions du CPM pendant qu'il était fonctionnel. Pour les FIFO du CPM reliés aux FIFO des unités de traitement, un changement de liens durant un traitement ne fait perdre aucune donnée. Par contre, si le FIFO n'était pas vide lors du changement de liens, celui-ci conserve ses données qui seront transférées lorsqu'une autre commutation de liens surviendra. Plusieurs problèmes ont été rencontrés relativement à l'horloge de transfert du CPM. Le CPM possède deux horloges, une pour le transfert de données entre ses FIFO et ceux des unités de traitement et une pour tout ce qui concerne son contrôle interne. La première est fournie par l'un des générateurs d'horloge du CSU et la seconde est fixe à 50 MHz et est la même que celle qui alimente le CSU. L'horloge fournie par les générateurs d'horloge du CSU est fixe à 8.33 MHz durant tout le temps que la carte est en mode configuration. Par contre, une fois que celle-ci passe en mode opérationnel, le CSU fournit l'horloge demandée par l'usager selon les commandes

envoyées par les ports de communication. Normalement, lorsque le CSU demande au CPM de fixer ses liens internes à une certaine configuration, celui-ci est encore dans le mode configuration et possède une horloge pour ces FIFO de 8.33 MHz. Par contre, lorsque nous voulons intervenir pendant que la carte est opérationnelle, l'horloge des FIFO du CPM n'est pas nécessairement à 8.33 MHz, puisqu'elle est à la fréquence fixée par l'usager. Ainsi, selon des simulations avec délais que nous avons effectuées, lorsqu'on ordonne au CPM de changer ces liens, la commande est exécutée environ une fois sur deux, si la fréquence est autre que 8.33 MHz.

Ces faits sont facilement explicables. Lorsque la carte est en mode configuration, le CSU fournit une horloge de 8.33 MHz qui est synchronisée avec son horloge interne à la même fréquence. Selon le design d'origine, le protocole de communication entre le CPM et le CSU est basé sur le fait que toutes les commandes transmises du CSU au CPM opèrent à 8.33 MHz dans le mode de configuration de la carte. Ainsi, lorsque la fréquence est différente, particulièrement lorsqu'elle est plus rapide, les délais internes au CPM ne sont plus respectés, ce qui résulte en un blocage de tous les transferts futurs du CPM puisque celui-ci ne répond plus.

Donc, outre la commande de changement des liens internes du CPM, toutes les instructions JTAG implantées dans le CSU fonctionnent, mais elles utilisent également un pourcentage très élevé de la surface de silicium disponible sur le FPGA. Ainsi, l'implantation matérielle de ce projet s'est arrêtée à la banque d'instructions présentée au chapitre deux. En effet, le contrôle de la mémoire DRAM que nous voulions implanter nécessitait beaucoup trop de place, puisque c'est au moins deux machines à états synchronisées entre elles qu'il faudrait implanter, soit une pour les rafraîchissements de la mémoire et l'autre pour tous les transferts désirés. Pour ce qui est du contrôle des liens Ilink, présents entre les unités de traitement et le CSU, c'est la même chose. Implanter un système capable de gérer l'entrée et la sortie de données sur 34 lignes de données nécessite trop de place. Finalement, pour ce qui est de la commande de changement d'horloge, elle aurait pu être implantée puisqu'elle ne nécessite pas beaucoup de logique, mais étant donné l'application que nous avons réalisée, celle-ci s'avérait inutile.

3.2 L'IMPLANTATION LOGICIELLE

3.2.1 NOTIONS DE BASE

L'implantation logicielle dépend de plusieurs facteurs que nous n'avons pas encore introduits dans ce mémoire. Cette partie se situe à un plus haut niveau d'abstraction que la partie matérielle, puisque c'est l'aspect logiciel qui contrôle ce système basé sur la technologie des DSP de Texas Instruments TMS320C40. Quatre principaux éléments font partie du système global. On retrouve en premier lieu le processeur principal; un DSP C40, la carte X-C436 qui agit à titre de co-processeur pour le DSP et un TBC (Test Bus Controller de Texas Instruments) ou tout autre module capable de sérialiser les données selon le protocole JTAG. Le quatrième élément du système et non le moindre est le système hôte duquel dépend toute l'architecture. C'est ce système de commande qui permettra de supporter la reconfiguration dynamique puisqu'il interviendra auprès du TBC et du DSP à titre d'intermédiaire de communication. De plus, le système hôte est le seul à posséder l'accès à la bibliothèque de configurations qui prendront place successivement dans les unités de traitement, puisque nous avons convenu de transférer ces fichiers de configuration par l'intermédiaire du lien JTAG.

En plus de considérer ces modules principaux, le système varie selon l'application implantée. Il se peut que d'autres modules se greffent au système et ils peuvent représenter un autre dispositif à contrôler pour le système hôte. Des modules supplémentaires peuvent également être des sources de données ou d'autres co-processeurs coopérant avec la carte X-C436. Toutefois, les quatre modules présentés plus haut représentent la partie fixe de ce système, que l'on retrouvera indépendamment de l'application implantée. De cette manière, nous avons créé des commandes haut niveau qui permettent d'utiliser la reconfiguration dynamique avec la carte X-C436 en harmonie avec la bibliothèque de fonctions qu'elle possédait déjà lors de sa commercialisation et qui permettait de l'utiliser avec un système basé sur un DSP C40 de Texas Instruments. Les commandes rajoutées sont cette fois au niveau du système hôte (code en C). Elles permettent d'établir la communication avec le CSU une

fois que la carte est en traitement, et ainsi enclencher le processus de reconfiguration d'une des unités de traitement. La bibliothèque de fonctions existantes était, quant à elle, codée pour le DSP et l'usager n'avait accès qu'au fichier objet seulement. Les nouvelles commandes implantées sont fixes mais elles doivent être adaptées à plus bas niveau pour le système sur lequel l'application sera implantée. En effet, pour chaque contrôleur TBC existant, différentes manières d'adressage existent, ainsi qu'un nombre différent de registres. C'est ce type d'adaptation que l'usager doit être en mesure de faire.

L'application en tant que telle fournit le mode de communication que devra adopter le système hôte et toutes les particularités de contrôle auxquelles il devra faire face. Par exemple, le DSP du système peut être configuré à l'aide d'une mémoire PROM qu'il possède ou encore être configuré via ses ports de communication. Dans ce dernier cas, c'est le système hôte qui doit alimenter les ports de communication du DSP avec son fichier de configuration. De la même manière, les commandes que la carte X-C436 reçoit lorsqu'elle est en mode configuration doivent respecter un certain protocole et sa configuration est habituellement prise en charge par le DSP une fois que celui-ci est lui-même configuré. Les commandes d'initialisation d'une carte X-C436 ont été développées chez MiroTech et elles font partie du logiciel vendu avec la carte. Il se peut cependant que le système hôte doive se charger de la configuration de la carte, cela dépend de l'application implantée et des besoins de l'usager. C'est donc ce genre de particularités qui peuvent être variables d'un système à l'autre, tout comme le nombre d'éléments dans le système.

3.2.2 L'APPLICATION IMPLANTÉE: SUITE DE FILTRES SOBEL

Afin de tester le système et de mesurer ses performances, nous avons implanté une application qui procédait en une série de convolutions sur une image défilante, telle l'image captée par une caméra ou projetée par un vidéo. En appliquant une série de filtres légèrement différents entre eux, on pourrait voir l'évolution par exemple d'un Sobel rotatif. Un filtre Sobel permet de mettre en relief les contours verticaux ou

horizontaux d'une image selon sa configuration. Pour un convolueur 3x3, la matrice de poids peut être représentée selon une des deux matrices de poids qui suivent:

$$matriceA = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$ Sobel Vertical

$$matriceB = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$ Sobel Horizontal

En pondérant la combinaison de deux Sobel à la fois, un vertical et l'autre horizontal, on peut, en appliquant successivement le résultat de ces combinaisons, obtenir l'effet d'un Sobel mettant en relief les contours de manière rotative. Cette application permettrait de mesurer le temps que prend le système à effectuer la reconfiguration d'une unité de traitement, puisque pour chaque configuration de Sobel appliquée, on effectue une alternance dans le traitement et la reconfiguration d'une unité de traitement. L'équation ci-dessous représente la forme générale de chaque matrice de convolution qui a été appliquée sur l'image défilante. Pour les fins de cette application, nous n'avons construit que 64 filtres différents à partir de cette équation.

$$\sum_{x=1}^{x=3}\sum_{y=1}^{y=3} matriceT_{[x][y]} = \sin\phi \sum_{x=1}^{x=3}\sum_{y=1}^{y=3} matriceA_{[x][y]} + \cos\phi \sum_{x=1}^{x=3}\sum_{y=1}^{y=3} matriceB_{[x][y]}$$

Où ϕ est l'angle résultant selon lequel la détection de contour est marquée. Noter également que matriceT, est la matrice de transformation résultant qui est appliquée sur l'image traitée et que matriceA et matriceB sont les matrices présentées plus haut.

3.2.2.1 Mécanisme de reconfiguration

La reconfiguration d'une unité de traitement nécessite le transfert d'un fichier de configuration de 830 kbit et ce, transféré de manière sérielle et à une vitesse maximale de 10 MHz (transfert à 10 Mbit/s). Le CSU a été conçu pour configurer les

unités de traitement de manière sérielle seulement, mais les FPGA de la famille XC4000 possèdent aussi un mode de configuration parallèle, qui permet de multiplier d'un facteur de 8 la vitesse maximale de transfert et d'obtenir un taux de transfert de 80 Mbit/s. Cependant, puisque les liens physiques de l'unité de traitement nécessaires pour effectuer une configuration selon le mode parallèle ne sont pas reliés au CSU et que nous ne pouvons pas modifier l'architecture de la carte, nous devons utiliser le mode de configuration sériel.

Ainsi, de manière théorique, possédant une horloge JTAG à 10 MHz, nous pourrions effectuer une reconfiguration en 83 milli-secondes. En incluant tous les temps requis lors des communications entre le TBC et le CSU, ainsi que le temps nécessaire pour effectuer une remise à zéro de la configuration du FPGA, le temps qui devrait être alloué pour effectuer une complète permutation du traitement entre deux filtres pourrait tourner autour de 100 ms au maximum. En effet, puisque l'intervention auprès du TBC est effectuée à l'aide du système hôte et puisque celui-ci est habituellement un micro-ordinateur dont le système d'exploitation est Windows, la tâche requise sur le TBC doit être partagée avec les autres tâches qu'il doit effectuer. Ainsi, certaines latences, aléatoirement disposées, peuvent être introduites sans que nous en ayons le contrôle. Donc, un temps global pour passer d'une configuration à l'autre autour de 100 ms semble être un estimé raisonnable, selon le système que nous possédons. Nous pouvons donc mesurer qu'environ trois images seront traitées par configuration de filtre, puisque le flot d'images nous procure un taux de 30 images par seconde.

3.2.2.2 Déroulement du programme implanté

L'implantation réalisée nécessite, en plus des éléments de base dont nous avons parlé plus haut, d'un *Frame Graber* qui nous permet de capter l'image d'un signal vidéo ou d'une caméra et de la traiter. Le *Frame Graber* empaquette l'information de chrominance et de luminance selon ce qu'on lui demande et sous la forme qu'on lui demande. Les filtres de convolution ne sont effectifs que sur la partie

luminance de l'image. En utilisant un convolueur 3x3, chaque pixel est traité selon le poids accordé à ses huit voisins immédiats et à lui-même.

La figure 3.2 représente le schéma bloc du système implanté. On peut voir que le *Frame Graber* utilisé possédait également un DSP TMS320C40 de Texas Instruments en plus des mécanismes de capture de l'image proprement dits. Ce processeur se charge de rafraîchir et de lire la mémoire vidéo en plus d'y écrire l'image captée. Pour traiter l'image, il transmet à la carte X-C436 l'information de luminance de l'image captée et il reprend de la même carte le résultat du filtre. Le DSP présent sur le *Frame Graber* est un élément important dans la reconfiguration dynamique, puisque c'est lui qui se charge d'effectuer le transfert de traitement. En effet, on peut remarquer qu'entre la carte X-C436 et le *Frame Graber*, il y a deux liens entrants ainsi que deux liens sortants. Ces liens sont les ports de communication qui se chargent de

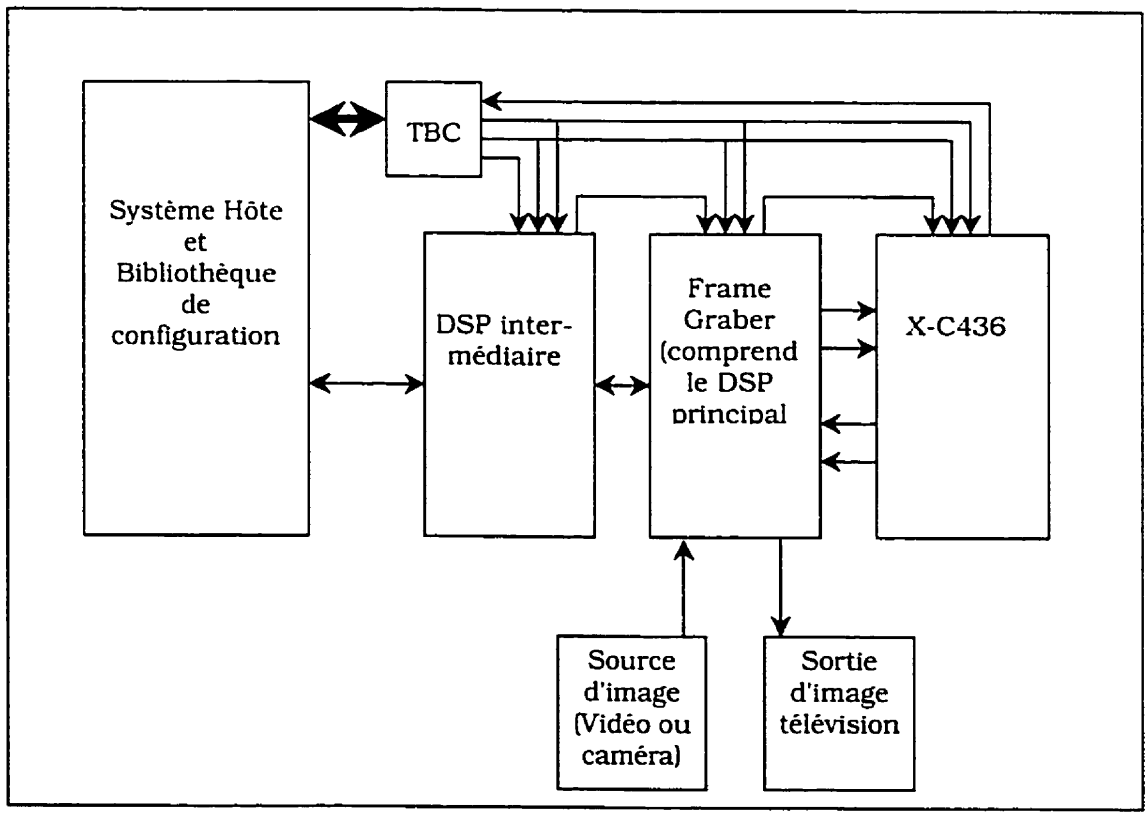

Figure 3.2 Schéma bloc du système avec l'application de suite de filtres d'image

transporter les données. Ainsi, de manière fixe, les deux unités de traitement sont reliées à la source de données, soit le *Frame Graber*, mais ne reçoivent des données à traiter que lorsqu'ils sont configurés pour le faire et effectuent ainsi le traitement à tour de rôle. Pour ce qui est de la permutation de traitement en tant que tel, le système hôte transmet une commande de permutation au DSP intermédiaire qui, à son tour, la redirige vers le *Frame Graber*. Celui-ci attend qu'une image se termine avant de permuter la sortie des données pour ne pas couper les images. Autrement dit, un seul flot de données est fonctionnel à la fois.

Pour le retour de l'information, aucune mesure de la sorte n'est nécessaire, car comme les unités de traitement ne peuvent recevoir que des images entières, le traitement est toujours effectué sur une image complète. Ainsi, les deux ports de communication qui ramènent les données de la carte X-C436 au *Frame Graber* possèdent chacun une DMA (Direct Memory Address) de transfert simultané qui redirige le contenu de ces deux ports de communication au même espace mémoire. Il est impossible que cette opération provoque une incompatibilité dans l'écriture pour les raisons nommées plus haut.

La figure 3.3 illustre le flot de données du traitement d'une image dans le DSP du *Frame Graber*. La DMA 1 effectue le transfert des données arrivant d'un vidéo ou d'une caméra vers un espace précis dans la RAM Vidéo. La DMA 2 effectue la

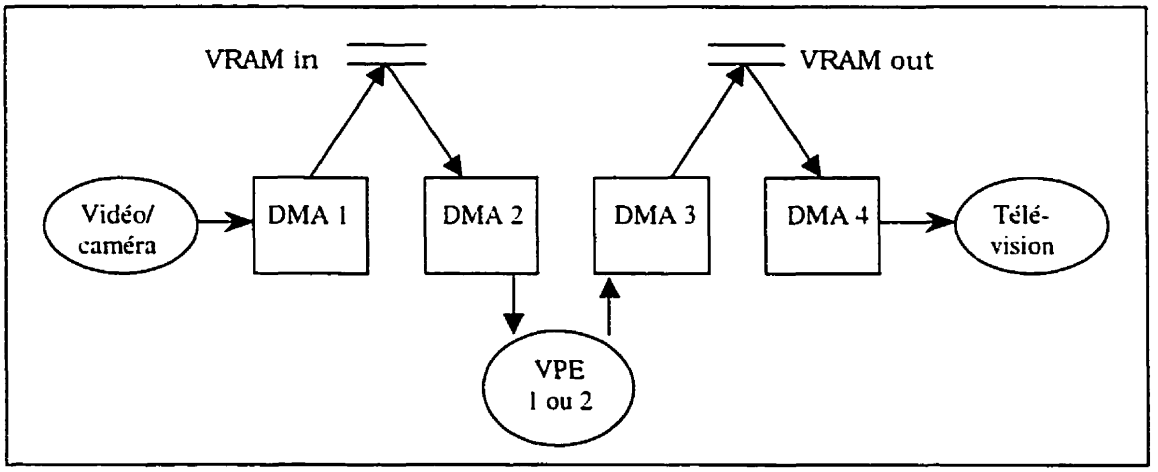

Figure 3.3 Flot de données du traitement d'image

translation des données de la VRAM vers le port de communication du VPE traitant. La routine qui effectue cette DMA est interrompue à la fin d'une image si une demande de changement de VPE traitant est envoyée. L'adresse de sortie des données fait seulement passer d'un port de communication à un autre et le transfert des données d'image n'est jamais ralenti par ce changement. La DMA 3 quant à elle est synchronisée sur l'entrée de deux ports de communication à la fois, soit la sortie des deux unités de traitement. Les données qui arrivent de l'un ou de l'autre sont transférées à la même espace mémoire, c'est-à-dire celle réservée à la mémoire VRAM de sortie. Finalement, la DMA 4 se charge de pousser les données de la VRAM de sortie vers l'affichage.

La commande que retransmet le DSP intermédiaire au *Frame Graber* provoque une suite d'événements chez ce dernier. En effet, pour assurer la plus grande sécurité dans ce mode de permutation, le *Frame Graber* doit retourner un mot qui indique au DSP intermédiaire qu'il a exécuté la commande demandée. Le DSP intermédiaire retransmet cette information à la carte. Pour la carte, cette réponse du *Frame Graber* enclenche le processus de reconfiguration de l'unité de traitement qui ne travaille plus.

Il peut sembler inadéquat de posséder un DSP intermédiaire dédié à retransmettre les commandes du système hôte au DSP du *Frame Graber*, mais sa présence est surtout due à une question de circonstance. En effet, il était nécessaire de posséder une plate-forme pour installer toutes ces cartes et cette plate-forme devait également tenir le rôle de lien entre le système hôte et le reste des cartes. Nous avons utilisé une carte de la compagnie Sundance qui possédait quatre sites TIM en plus d'un TBC. Cette carte peut être installée (quoique très volumineuse) à l'intérieur d'un ordinateur personnel à l'aide de son bus PCI (33 MHz, 32 bits). Les composants accessible à partir du système hôte, en plus de quelques registres de configuration, sont le TBC et un port de communication (numéro 3) du DSP du site TIM A.

Cependant, cette carte possède certaines restrictions au niveau des liens entre les différents sites. Par exemple, seule la carte située sur le site A possède un port de communication et des accès mémoire avec le système hôte. De plus, les sites possèdent certains ports de communication connectés de manière fixe entre eux, à la manière d'une chaîne, ce qui alloue moins de flexibilité pour établir l'architecture des

cartes. Finalement, le site A comporte un site TIM particulier qui double un de ses connecteurs pour accepter un bus global. Normalement, une carte ne possède pas de bus global, mais plutôt des connecteurs conventionnels qui sont au nombre de deux, l'un au nord de la carte et l'autre au sud. Par contre, une carte conventionnelle peut habituellement tenir sur un site où un connecteur supplémentaire pour le bus global est présent. Ce qui n'est pas le cas avec le *Frame Graber*. La physionomie de celui-ci est particulière, puisqu'il a la dimension de deux cartes TIM et il possède quatre connecteurs. La communication d'une des paires de connecteurs est réservée au DSP de la carte, tandis que l'autre est plutôt vacante et ne possède pas de signaux que nous utilisons. De plus, le *Frame Graber* est lourdement chargé de composants et il est physiquement impossible de connecter la carte au site A qui est munie d'un connecteur supplémentaire, puisque l'endroit où le connecteur du bus global devrait se trouver est rempli de composants. Il n'est donc pas complètement conforme aux spécifications de Texas Instruments pour ce qui est des contraintes mécaniques de ce connecteur. Ainsi, au risque de ralentir légèrement le système de communication entre le système hôte et le DSP du *Frame Graber*, nous avons installé un intermédiaire de communication.

Par contre, puisque le système hôte ne possède pas de lien direct avec le *Frame Graber*, le DSP intermédiaire se charge, une fois qu'il est lui même fonctionnel, de programmer le DSP du *Frame Graber*. Dans son programme, le DSP du *Frame Graber* possède les deux premières configurations qui prennent place à l'intérieur des unités de traitement, ainsi que toutes les commandes de mise en marche de la carte X-C436. Par la suite, le système hôte se charge de commander toutes les permutations et changements de configuration des unités de traitement. Les trois algorithmes qui suivent représentent le code qui est exécuté dans chacun des trois éléments logiciel de ce système, soit le système hôte, le DSP principal et le DSP secondaire.

Algorithme du programme principal exécuté par le système hôte

Auteur: Cynthia Cousineau,
Auteur code interface TBC: Martin Filteault

```
1  Initialisation de la carte Sundance et du TBC de la carte
2  Envoi du programme du DSP intermédiaire
3  Envoi du programme du DSP principal (Frame Graber)
4        SELON le choix de l'usager
5        =Commande BYPASS
6              Envoi du mot de commande BYPASS au Frame Graber
7              Attente de la réponse du Frame Graber (Acknowledge)
8  -----  =Commande BYPASS OFF
9              Envoi du mot de commande BYPASS OFF au Frame Graber
10             Attente de la réponse du Frame Graber (Acknowledge)
11 -----  =Commande HALF BYPASS
12             Envoi du mot de commande HALF BYPASS au Frame Graber
13 -----  =Commande FREEZE
14             Envoi du mot de commande FREEZE au Frame Graber
15             Attente de la réponse du Frame Graber (Acknowledge)
16 -----  =Commande FREEZE OFF
17             Envoi du mot de commande FREEZE OFF au Frame Graber
18             Attente de la réponse du Frame Graber (Acknowledge)
19 -----  =Commande SUITE SOBEL
20             *  Pour les 64 configurations de filtres
21                Reconfiguration du VPE qui ne traite pas
22                Envoi de la commande VPE1 ou VPE2 indiquant un
                  changement de VPE traitant au DSP principal (via le DSP
                  intermédiaire)
23                Attente de la réponse du DSP principal via DSP intermédaire
```

Algorithme du programme principal exécuté par le DSP intermédiaire

Auteur: Cynthia Cousineau

```
1  Programmation du DSP principal
2        RÉPÉTER
3              Redirection des données provenant du système hôte vers le
```

```
                DSP principal
4               Redirection des données provenant du DSP principal vers le
                système hôte
5       *       JUSQU'À fin programme
```

Algorithme du programme principal exécuté par le DSP principal

Auteur: Martin Filteault
Modifié par: Cynthia Cousineau

```
1   Initialisation de la mémoire VRAM
2   Configuration initiale de la carte X-C436 et de ses VPE
    Départ des DMAs de transfert des deux espaces mémoire VRAM
3   Départ de la routine de transfert des données d'image au VPE traitant
4   Départ de la DMA de transfert des données traitées vers la mémoire VRAM de
    sortie
5   RÉPÉTER
6       SELON le mot de commande
7       =Commande BYPASS
8           Variable MODE devient BYPASS
9           Appeler la routine d'affichage en lui passant la variable
            MODE
10  -----   =Commande BYPASS OFF
11          Variable MODE devient BYPASS OFF
12          Appeler la routine d'affichage en lui passant la variable
            MODE
13  -----   =Commande HALF BYPASS
14          Variable MODE devient HALF BYPASS
15          Appeler la routine d'affichage en lui passant la variable
            MODE
16  -----   =Commande FREEZE
17          Variable MODE devient FREEZE
18          Appeler la routine d'affichage en lui passant la variable
```

				MODE
19			-----	=Commande FREEZE OFF
20				Variable MODE devient FREEZE OFF
21				Appeler la routine d'affichage en lui passant la variable MODE
22			-----	=Commande VPE1 ou VPE2
23				Variable COMPORT devient NOUVEAU VPE
24				Appeler la routine d'affichage en lui passant la variable COMPORT
25		*		JUSQU'À fin du programme

Note: Lorsque la routine d'affichage est appelée, deux variables sont transférées mais une seule des deux change à la fois, c'est pourquoi, dans le précédent algorithme, nous indiquons que la fonction est appelée et qu'une seule variable est passée.

3.2.3 PROCESSUS DE RECONFIGURATION DYNAMIQUE

Une fois que la carte est initialement configurée, le processus cyclique de reconfiguration dynamique peut s'enclencher. D'abord, le système hôte envoie, via le TBC, une commande au CSU de la carte X-C436 pour lui indiquer de faire une remise à zéro de la configuration de l'unité de traitement disponible (celle qui ne traite pas). Pour ce faire, il transmet au CSU l'instruction JTAG "CFGINFO", qui contient l'information sur l'unité de traitement qui sera reconfigurée, ainsi qu'une constante relative à la configuration en elle-même, dont nous parlerons plus loin. Par la suite, après avoir envoyé une commande pour lire le statut des lignes de configuration de l'unité de traitement concernée (commande STATUT) et vérifié que l'unité de traitement était en mesure de se faire configurer, le système hôte envoie l'instruction "CFGDATA" qui contient les données de configuration. La constante envoyée lors de la première instruction de remise à zéro est nécessaire pour introduire un délai avant la transmission des vraies données de configuration.

En effet, nous savons que le CSU peut être placé à n'importe quel endroit dans une chaîne JTAG. Nous savons également que l'horloge TCK, ainsi que le signal TMS, sont envoyés à tous les composants de la chaîne de manière parallèle. Ceci implique que toutes les machines à états de tous les composants de la chaîne se retrouvent dans le même état en même temps, mais elles ne répondent pas à la même instruction. Cependant, les données en tant que telles parviennent tour a tour aux composants de manière sérielle. Ainsi, si la carte X-C436 est placée après un autre composant, même si ce dernier est en mode BYPASS, un délai d'un cycle d'horloge est tout de même nécessaire avant que la première donnée valide lui parvienne. Si le composant devant la carte n'est pas en mode BYPASS, mais plutôt dans une autre instruction, plus de cycles encore sépareront les premiers coups d'horloge de la première donnée valide. Ainsi, il est nécessaire de placer un compteur réglable de manière logicielle qui permet d'ajuster le délai propre à la configuration du système et la position de la carte X-C436 dans celui-ci. Une fois la valeur du délai passée, le CSU sait que les données qui lui parviennent sont les données faisant partie du fichier de configuration du FPGA visé.

Comme tout le contrôle a été ajusté de manière logicielle, le CSU n'a pas à savoir si le fichier de configuration a été transmis en entier ou non. Il arrête de transmettre les données lorsque les signaux de contrôle JTAG, en occurrence TMS, lui indiquent de changer d'état. Ainsi, le système hôte peut contrôler la dernière étape nécessaire pour achever la mise en marche du FPGA en stimulant durant quelques cycles encore l'horloge de configuration du FPGA. Ceci s'appelle l'étape START-UP chez les FPGA Xilinx.

Une fois que le CSU a terminé la mise en marche du FPGA, celui-ci ne traite pas immédiatement des données, puisque le DSP du *Frame Graber* n'a pas encore été informé du transfert. Le système hôte envoie donc pour ce faire une commande au *Frame Graber* par l'entremise du DSP intermédiaire et lorsqu'il a atteint la fin d'une image, il exécute la commande et change la destination des données relatives à l'image. Aussitôt après, il renvoie un mot qui indique l'accomplissement de sa commande et le système hôte, en la recevant, initie le processus de configuration de l'unité de traitement qui est maintenant libre et le cycle recommence.

Une image contient deux champs, un pair et un impair comptent chacun 240 lignes et de 640 pixels par lignes. Ces deux champs font partie d'une image entière qui contient, en plus de cette information, 45 lignes d'informations diverses, disposées au début de chaque champ. On nomme ces lignes le *horizontal blank*. On retrouve donc un flot de donnée de 60 champs par seconde, ou encore 30 images complètes par seconde. Les deux champs d'image sont traités par le convolueur. La figure 3.4 montre, sans être à l'échelle, le temps requis pour chacune des étapes du cycle de reconfiguration dynamique. On peut remarquer que l'étape de la permutation des unités de traitement possède un temps d'exécution variable, étant donné que le *Frame Graber* n'exécute la commande que lorsque le champ d'image qu'il est en train d'envoyer est terminé. Le temps de transfert d'un champ entier est d'environ 16.6 ms et c'est donc le temps maximal auquel nous pouvons nous attendre pour cette étape. Par contre, étant donné les capacités de traitement du DSP du *Frame Graber* ainsi que les capacités de traitement des convolueurs implantés dans les unités de traitement, ce n'est pas la taille entière du champ d'image qui sera traité. Le temps de traitement d'une image par les unités de traitement lorsque leur configuration répond à la forme d'un convolueur 3x3 est d'environ 7.7 ms par image de taille 240 lignes, 320 pixels par ligne. Le convolueur traite donc un pixel sur deux. L'architecture de ce convolueur a été développée préalablement chez MiroTech pour la carte X-C436 et leurs unités de traitement.

3.3 Résultats et Analyse

La série de filtres Sobel que nous avons implantée représentait la succession de 64 configurations de convolueur qui prenaient place dans l'une ou l'autre des unités de traitement. L'image tirée d'un vidéo ou d'une caméra était continuellement traitée et on pouvait remarquer la subtile différence entre chacun des filtres qui étaient appliqués. Nous avions mentionné que nous désirions obtenir un temps de reconfiguration autour de 100 ms et ce pour tout le cycle de reconfiguration. Malheureusement, après plusieurs tests et mesures, nous n'avons pas réussi à augmenter la fréquence de reconfiguration à plus de quatre reconfigurations à la seconde, alors que nous croyions

être en mesure d'atteindre jusqu'à dix reconfigurations à la seconde. En effet, le temps requis pour exécuter la suite des 64 reconfigurations prend en moyenne un peu plus de 14 secondes.

Nous avons séparé les étapes requises pour effectuer un cycle de reconfiguration et nous avons mesuré le temps requis pour chacune de ces étapes afin de déterminer lequel causait ce retard. Tous les temps que nous avons mentionnés dans l'échelle temporelle de la figure 3.4 étaient véridiques, à l'exception du temps requis pour effectuer la reconfiguration proprement dite. L'horloge du TBC qui était fournie avec la plate-forme était fixe à 8.25 MHz. En effet, l'architecture de cette plate-forme faisait en sorte que le TBC se procurait toujours une horloge dérivée de celle du bus PCI de la carte, plus précisément le quart de celle-ci. Ainsi, puisque le micro ordinateur que nous utilisions fournissait une horloge de 33 MHz pour le bus PCI, nous retrouvions une fréquence de 8.25 MHz pour l'horloge TCK du TBC. La fréquence de cette horloge est un peu plus lente que celle que nous croyions utiliser et à partir de laquelle nous avons établi les temps montrés à la figure 3.4, à savoir 10 MHz. Cependant, même en considérant l'envoi du fichier de configuration de la taille 830 kbits à 8.25 MHz, nous devrions obtenir un temps d'envoi de 100.6 ms au lieu de 83 ms comme nous l'avions indiqué. Or ceci n'explique pas le fait qu'un cycle de reconfiguration prend entre 220 et 230 ms à s'exécuter.

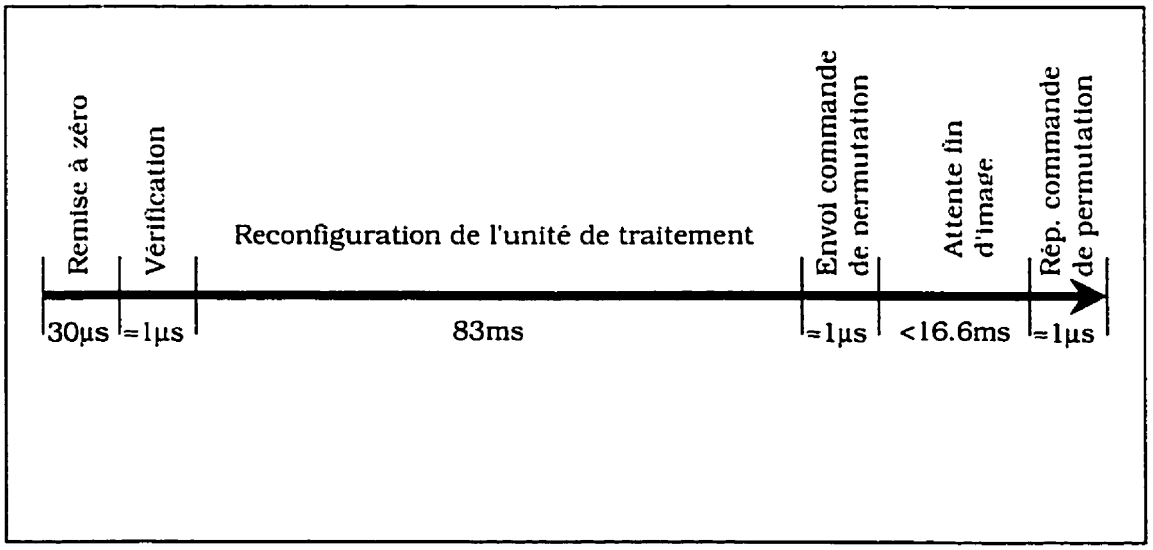

Figure 3.4 Représentation temporelle d'un cycle de reconfiguration dynamique

Nous avons finalement conclu que le temps de reconfiguration était plus long que prévu à cause du temps que prend le TBC à recharger ses FIFO de données lors du transfert du fichier de reconfiguration. En effet, comme la longueur des FIFO du TBC est limitée, celui-ci place les éléments de la chaîne JTAG dans l'état PAUSE avant de poursuivre son transfert. Ce qui explique que le temps de reconfiguration soit environ le double du temps que nous avions prévu.

Les performances que nous voulions obtenir, soit 100 ms pour un cycle de reconfiguration, n'ont pas été atteintes, mais le résultat obtenu n'est pas pour autant décevant. En effet, pour pouvoir effectuer du traitement temps réel, ce sont des performances dans l'ordre de la micro seconde qu'il aurait fallu obtenir et nous savions dès le départ que c'était hors de notre portée. Pour obtenir des temps de reconfiguration aussi petits, nous aurions dû utiliser soit des FPGA très petits en superficie ou encore des FPGA partiellement reconfigurables. Mais là encore, ce n'est pas ce que nous cherchions, puisque nous désirions des FPGA suffisamment gros pour pouvoir y implanter des systèmes viables, fonctionnels et réalistes. De plus, même si notre système n'est pas en mesure d'être reconfiguré en deçà de 230 ms, nous n'avons tout de même aucun point mort dans le système. Celui-ci continue toujours le traitement contrairement à ce que peut offrir un système muni d'un FPGA reconfigurable partiellement.

Quelques solutions ont été envisagées pour permettre de reconfigurer le système dans les temps. La première est de ne pas utiliser le TBC pour transmettre la reconfiguration, mais d'utiliser la mémoire DRAM qui est sur la carte et accessible par le CSU. Celle-ci possède une bande passante largement suffisante pour la transmission de la reconfiguration d'un FPGA de la famille XC4000 et n'introduirait pas de pause augmentant le temps de transmission du fichier de configuration. Malheureusement, comme nous l'avons mentionné, le FPGA contenant le CSU est déjà rempli à plus de 80 % avec les commandes de base. Nous n'avons pas abordé ce développement, parce qu'il nécessitait l'opération de coupures dans les fonctions que le CSU possède déjà, ainsi qu'un travail très pénible en perspective pour accomplir le placement/routage d'un FPGA trop rempli.

Une autre solution envisageable serait de développer un composant qui répondrait aux mêmes fonctions qu'un TBC, mais spécialement conçu pour avoir un flot de transmission continu, lui permettant de transmettre le fichier de configuration d'un FPGA sans interruption. Nous n'avons pas expérimenté cette solution, puisque même si, à prime abord, elle semble convenir à nos besoins, elle peut s'avérer très compliquée à implanter. En effet, développer un tel système ne devrait pas poser un trop gros problème en soi, mais l'interface et la forme physique de ce nouveau composant peut, quant à lui, nous donner du fil à retordre. Il faudrait trouver une plate-forme quelconque, capable de s'adapter à tous les types de système. Il ne faudrait évidemment pas que cette solution n'en vienne à introduire une nouvelle carte, parce qu'ainsi, tous les bénéfices de la reconfiguration dynamique et de son économie matérielle tomberaient à l'eau. Il peut donc être plus nocif que bénéfique d'introduire un dispositif améliorant les performances du système, afin de baisser d'un facteur de 2 à 3 le temps que prend celui-ci à se reconfigurer.

3.4 Conclusion

Ce chapitre nous a permis d'exposer plus profondément les concepts et les mécanismes mis en branle dans le développement de ce projet de maîtrise. Nous avons exploré les spécifications matérielles et logicielles du projet, en plus de spécifier les besoins du système lorsqu'une application est implantée sur celui-ci. Nous avons également tenté de trouver les lacunes du système et nous avons proposé des solutions pouvant mener à une amélioration des performances. Il est clair que ces solutions ne sont pas la clé qui résout les problèmes reliés à notre implantation et c'est pourquoi nous discuterons plus en détail des performances de notre système, de ses forces et de ses faiblesses au Chapitre 4.

CHAPITRE QUATRIÈME - DISCUSSION

À la fin du dernier chapitre, nous avons abordé comme principal point à améliorer le temps de reconfiguration de notre système. Nous avons également mentionné que peu importe la méthode choisie pour atteindre le temps minimal de reconfiguration d'une unité de traitement, soit de 100 ms, nous ne pourrons jamais traiter des applications temps réel qui nécessitent un temps de reconfiguration autour de quelques micro secondes. Cependant, il est clair que notre système peut être beaucoup plus avantageux que ceux utilisant les FPGA reconfigurables dynamiquement, puisqu'aucun temps d'arrêt n'est nécessaire pour passer d'une configuration à l'autre, contrairement aux autres systèmes. De plus, notre système est une preuve réelle et fonctionnelle de reconfiguration dynamique et aucune autre carte présentement sur le marché n'offre de telles possibilités.

Que pouvons-nous faire dans ce cas pour réduire le temps de reconfiguration de notre système? Pouvons-nous porter le concept à des technologies plus avancées de FPGA tels les nouveaux Virtex? Bien entendu, si la possibilité de construire un système entièrement dédié à la reconfiguration dynamique s'offrait à nous, le système qui serait alors conçu aurait une architecture différente de la carte X-C436. En effet, dans les cartes conventionnelles utilisant des modules de logique programmable comme des FPGA pour agir à titre de coprocesseur, il est souvent de mise de penser que le temps de configuration d'une unité importe peu, puisque cette configuration s'effectue avant même de commencer le traitement. Lorsqu'on conçoit l'architecture de telle carte, on mise d'abord sur la rapidité du transfert de données et sur l'architecture des configurations qui effectueront le traitement, afin que celle-ci soit en mesure de sortir les données traitées le plus rapidement possible et avec le moins de contraintes possibles.

Pour qu'une carte ait un temps de reconfiguration performant, il faut d'abord utiliser l'accès de configuration le plus rapide qu'elle possède. Par exemple, la plupart des FPGA possèdent un mode de configuration parallèle qui permet de diminuer le temps de reconfiguration d'un facteur de huit environ. Étant donné que les FPGA ont

tendance à devenir très gros en nombre de portes logiques, les fichiers de configuration vont dans le même sens et passent à des tailles de 5.8 Mbits (735.8 kBytes) pour un XCV1000, le plus gros des Virtex présentement disponible, comparativement à 813 kbits (101.6 kBytes) pour le FPGA utilisé comme unité de traitement sur la carte X-C436, soit un XC4036. Si la même fréquence d'horloge était utilisée pour configurer le XCV1000 que celle requise pour les FPGA de la famille XC4000, ce serait un temps de près de trois quarts de seconde qu'il faudrait mettre pour configurer ce FPGA de manière sérielle.

Par chance, les FPGA de la famille Virtex peuvent être configurés à une vitesse plus rapide que les FPGA de la famille XC4000. D'après les données fournies par Xilinx dans la feuille de spécification du Virtex, la vitesse maximale à laquelle pourrait être stimulée l'horloge de configuration CCLK est de 66 MHz. Si on choisit de configurer le FPGA par le mode 8 bits parallèle, nommé SelectMap pour le Virtex, on pourrait configurer le FGPA le plus volumineux du marché présentement en 11.6 ms, ce qui est comparable au temps minimal qu'on peut obtenir chez les FPGA de la famille XC4000.

Nous avions mentionné au premier chapitre que les principaux problèmes reliés à l'utilisation des FPGA reconfigurables partiellement étaient le manque d'outils de placement routage et d'outils nécessaires au développement d'une architecture ayant comme base la reconfiguration dynamique. De plus, la taille des FPGA reconfigurables partiellement qu'on retrouvait alors sur le marché ne correspondait pas au besoin des architectures industrielles. Nous avons d'abord abordé dans cette discussion la possibilité de porter le concept développé sur la carte X-C436 sur un tout nouveau système, conçu à la base pour supporter la reconfiguration dynamique et ayant comme unités de traitement des Virtex. Nous avons établi que pour ce qui est de la reconfiguration totale du FPGA, les performances de ce FPGA égalaient celles de la famille XC4000, puisque même si la taille du fichier à transférer était plus volumineux, la vitesse à laquelle on peut le transmettre a, quant à elle, augmentée. Le problème reste donc entier. Par contre, si on considère la fonctionnalité de reconfiguration partielle de ces FPGA, on pourrait alors atteindre plusieurs caractéristiques du système désiré, à savoir un temps de reconfiguration dans l'ordre des micro secondes en utilisant la reconfiguration partielle et une capacité d'implantation largement suffisante pour répondre au besoin des architectures d'aujourd'hui.

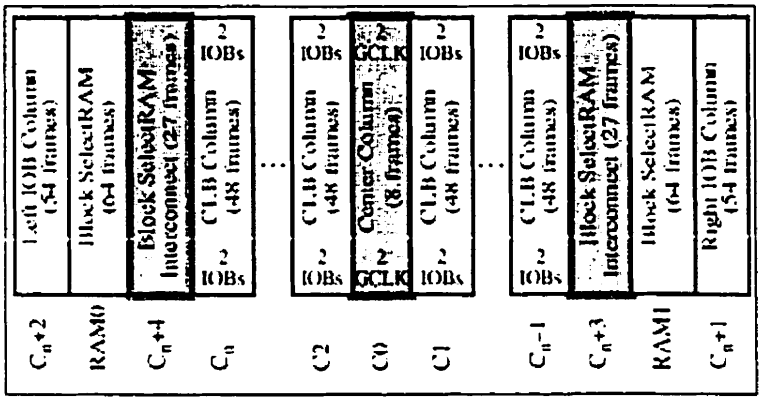

Figure 4.1 Représentation du nombre de blocs de configuration par colonne chez les FPGA Virtex de Xilinx. (Tirée de Xilinx Application Note: Virtex configuration architecture - Advanced users' guide, 1999, page 2.)

La reconfiguration partielle chez les Virtex de Xilinx n'inclut pas seulement la possibilité d'écrire sur certaines parties cibles du FPGA, mais aussi de relire ces parties. On peut ainsi être en mesure de connaître des résultats partiels d'un traitement en cours en relisant la valeur de certains signaux, ou certains espaces mémoire. Les modes de configuration disponibles pour cette famille de FPGA sont les modes sériels incluant, sériel-maître, sériel-esclave et par protocole JTAG, ainsi que le mode parallèle nommé SelectMap. Pour effectuer la relecture de la configuration, seulement les modes parallèle et sériel-JTAG peuvent être utilisés. Selon l'architecture du FPGA, on retrouve une organisation par colonne des CLBs et des blocs mémoire. Ainsi, une colonne comprend plusieurs CLBs qui eux-mêmes contiennent deux tranches (traduction de *slice*) ainsi que des blocs d'entrée/sortie à l'extrémité de la colonne. Le fichier de configuration, quant à lui, est formé de plusieurs blocs, qui formeront plus tard les unités de reconfiguration dynamique. Chaque colonne dans un FPGA nécessite, pour être configurée, ou lue, quarante-huit blocs du fichier de configuration. Pour les colonnes de cellules d'entrée/sortie aux extrémités gauche et droite du FPGA, elles nécessitent pour leur configuration cinquante-quatre blocs du fichier de configuration. De plus l'architecture du Virtex pourvoit celui-ci d'une certaine quantité de mémoire appelée Block SelectRAM et ceux-ci, pour leur configuration, nécessitent 64 blocs.

En publiant le processus de reconfiguration partielle et de relecture de la configuration, Xilinx permet de développer des systèmes utilisant la reconfiguration dynamique de manière très efficace, puisqu'on peut transmettre, d'une configuration à l'autre, des valeurs intermédiaires de traitement, avec la possibilité de relire le contenu du FPGA. De plus, cette dernière fonctionnalité permet de modifier adéquatement la nouvelle configuration, puisque la relecture de certaines valeurs critiques peut initier une reconfiguration, à l'image d'un système capable de s'auto-réguler.

Un seul problème persiste avec les Virtex, c'est le manque d'outils de placement routage et d'outil de développement. D'autres compagnies offrent des outils de synthèse pour les produits de Xilinx, comme par exemple Synplify de Synplicity ou FPGA express de Viewlogic. Ceux-ci supportent la synthèse VHDL ou Verilog de circuit développé pour les Virtex de Xilinx sans toutefois offrir un outil spécifique au développement de systèmes à base de reconfiguration dynamique. Quant à Xilinx, la société n'a pas annoncé à ce jour un logiciel de développement pour les systèmes utilisant la reconfiguration dynamique.

La reconfiguration dynamique est certes très utile pour certains systèmes, tels les réseaux de neurones et les systèmes de traitement de base de données comme l'ont démontré plusieurs études citées au chapitre 1. Par contre, nous ne verrons une vraie percée sur le marché de cette technique de reconfiguration que lorsque des outils performants seront disponibles commercialement. De plus, nous croyons que la reconfiguration dynamique possède un attrait certain, lorsque disponible sur une carte ou un système, puisqu'elle apporte une grande flexibilité à l'usager en ce qui a trait à la complexité des circuits concevables.

CONCLUSION

Le concept démontré au cours de ce mémoire est simple et portable. Il permet l'implantation de systèmes facilement développables contrairement à ceux qui utilisent des FPGA reconfigurables partiellement. De plus, les possibilités qu'offre l'utilisation du lien JTAG sur des systèmes mixtes (logiciel-matériel) ou sur des systèmes totalement matériels, sont aussi vastes que l'imagination du développeur. En effet, l'intégration de ce deuxième lien de communication permet de développer non seulement des systèmes reconfigurables dynamiquement, mais il permet également d'instaurer une série de commandes spécifiques qui peuvent conférer au système une grande flexibilité.

Bien entendu, il incombe au développeur de considérer avec attention tous les temps de traitement et de reconfiguration impliqués dans son système. Par contre, une fois que cette étude temporelle est faite, le développeur est en mesure de construire un système de reconfiguration dynamiquement efficace et économique puisqu'il aura considéré les capacités de son système et qu'il les aura exploitées. Nous devons également retenir que ce n'est pas tous les systèmes qui peuvent bénéficier de la reconfiguration dynamique puisqu'il faut d'abord que le système puisse se scinder en plusieurs traitements successifs. Ensuite, nous devons considérer seulement un système dont le temps d'exécution total de ces traitements soit d'un ordre de grandeur supérieur au temps de reconfiguration total du système. Cependant, en bout de ligne, pour les systèmes avantagés par la reconfiguration dynamique, l'économie au point de vue matérielle peut être importante.

Dans un avenir proche, les FPGA de plus grande taille seront les composants de base de plusieurs systèmes de pointe et il est fort probable que si la demande se fait sentir, des outils performants de placement routage ainsi que des outils de développement feront leur apparition sur le marché. Mais, d'ici là, le système qui a été développé dans le cadre de ce mémoire de maîtrise reste le seul à offrir présentement la possibilité d'implanter un système reconfigurable dynamiquement. Il est clair cependant que celui-ci pourrait présenter de meilleures performances s'il exploitait des FPGA reconfigurables partiellement, essentiellement grâce à leur modeste temps de

reconfiguration, mais encore faut-il être en mesure de développer de tels systèmes. En effet, si l'architecture proposée dans le cadre de ce mémoire était en mesure d'obtenir des temps de reconfiguration plus petit, c'est une plus vaste sélection d'application qui serait avantagée d'utiliser la reconfiguration dynamique. Les plus beaux exemples sont bien sur les systèmes temps réel.

L'essence même de ce projet était de développer un système reconfigurable dynamiquement et non seulement un FPGA seul et sans ressource. Nous avons enfin utilisé au maximum les ressources de reconfigurabilité que nous offrait cette famille de FPGA conventionnel, un aspect souvent inutilisé de ces FPGA. Nous croyons également que les recherches futures sur le sujet devraient proposer des solutions plus portés sur l'aspect global du problème de reconfiguration dynamique puisque l'on mise trop souvent, soit sur des systèmes dédiés dans les recherches académiques ou encore sur le FPGA seulement dans le cas de recherches industrielles. Nous croyons finalement que la reconfiguration dynamique peut encore être la source de projets très intéressants et très performants, mais beaucoup de travail au niveau des outils reste à venir.

BIBLIOGRAPHIE

ALBAHARNA, O. T., CHEUNG, P. Y. K., CLARKE, T. J. (1994). Area and Time Limitations of FPGA-based Virtual Hardware. International Conference on Computer Design, 184-189.

APEL, U. (1990). On-line Software Extension and Modification. Electrical Communication, volume 64, numéro 4. (Alcatel publication)

ARNOLD, J. M., BUELL, D. A., DAVIS, E. G. (juin 1992). Splash 2. Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 316-324.

BELLOWS, P., HUTCHINGS, B. L. (avril 1998). JHDL - an HDL for Reconfigurable Systems. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 175-184.

BITTNER, R. A. Jr., ATHANAS, P. M., MUSGROVEM. D. (1996). Colt: An Experiment in Wormhole Run-Time Reconfiguration. High-speed Computing Digital Signal Processing and Filtering Using Reconfigurable Logic, Boston, Massachusetts, États-Unis, 187-194.

BITTNER, R., ATHANAS, P. (1997). Wormhole Run-Time Reconfiguration. FPGA '97: ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis, 79-85.

BITTNER, R. A., ATHANAS, P. M. (avril 1997). Computing Kernels Implemented with a Wormhole RTR CCM. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 98-105.

BORRIELLO, G., EBELING, C., HAUCK, S., BURNS, S. (1995). The Triptych FPGA architecture. IEEE Transactions on Very Large Scale Integrated (VLSI) Systems, volume 3, numéro 4, 491-501.

BREBNER, G., GRAY, J. (1995). Use of Reconfigurability in Variable-lenght Code Detection at Video Rates. Field Programmable Logic and Applications, Oxford, Angleterre, 429-438.

BREBNER, G. (1996). A Virtual Hardware Operating System for the Xilinx XC6200. Proceeding of the 6th International Workshop on Field-Programmable Logic and Applications, Darmstadt, Allemagne, 327-336.

BREBNER, G. (1997). Automatic Identification of Swappable Logic Units in XC6200 Circuitry. Proceeding of the 7th International Workshop on Field-Programmable Logic and Applications, Londres, Angleterre, 173-182.

BREBNER, G. (avril 1997). The Swappable Logic Unit: a Paradigm for Virtual Hardware. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 77-86.

BREBNER, G., DONLIN, A. (1998). Runtime Reconfigurable Routing. Proceeding of the 8th International Workshop on Field-Programmable Logic and Applications, Tallinn, Estonie, 25-30.

BROWN, G. (novembre 1996). User-configurable Data Acquisition Systems. Proceedings of SPIE - The International Society for Optical Engineering High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic, Boston, Massachusetts, États-Unis, 54-64.

BURNS, J., DONLIN, A., HOGG, J., SINGH, S., de WIT, M. (avril 1997). A Dynamic Reconfiguration Run-Time System. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 66-75.

BUTTS, M. (mai 1995). Future Directions of Dynamically Reprogrammable Systems. Proceedings of the Custom Integrated circuits Conference, Santa Clara, Californie, États-Unis, 487-494.

CASSELMAN, S., THORNBURG, M., SCHEWEL, J. (1995). Hardware Object Programming on the EVC1: a Reconfigurable Computer. Proceedings FPGAs for Fast Rapid Board Development and Reconfigurable Computing, Photonics East, Philadelphie, Pennsylvanie, États-Unis, 168-176.

CADAMBI, S., WEENER, J., GOLDSTEIN, S. C., SCHMIT, H., THOMAS, D. E. (1998). Managing Pipeline-Reconfigurable FPGAs. Proceedings of the 1998 6th International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis , 55-64.

CARDOSO, J. M. P., NETO, H. C. (avril 1999). Macro-Based Hardware Compilation of Java™ Bytecodes into a Dynamic Reconfigurable Computing System. IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, Californie, États-Unis , 2-11.

CHANG, D., MAREK-SADOWSKA, M. (février 1997). Buffer Minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs. ACM/SIGDA International Symposium on Field Programmable gate Arrays - FPGA, Monterey, Californie, États-Unis, 142-148.

CHANG, D., MAREK-SADOWSKA, M. (1998). Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs. Proceedings of the 1998 6th International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis , 161-167.

CHEN, X.-Y., LING, X.-P., AMANO, H. (septembre 1994). Software Environment for WASMII - a Data Driven Machine with Virtual Hardware. Proceeding of the 4th International Workshop on Field Programmable Logic and Applications, Prague, République Tchèque, 208.

CHIRICESCU, S. M. S. A., VAI, M. M. (juin 1998). A Three-Dimensional FPGA with an Integrated Memory for In-Application Reconfiguration Data. Proceedings IEEE International Symposium on Circuits and System Processing of the 1998 ISCAS, Monterey, Californie, États-Unis , 232-235.

CHURCHER, S., KEAN, T., WILKIE, B. (septembre 1995). The XC6200 FastMap™ Processor Interface. Proceedings of the 5th International Workshop on Filed Programmable Logic and Applications, Oxford, Angleterre, 36-43.

COX, C. E., BLANZ, W. E. (mars 1992). GANGLION - A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier. IEEE Journal of Solid-State Circuits, volume 27, numéro 3, 288-298.

DAVIS, D., HARRIS, J. (1998). ACEcard: A High-Performance Architecture for Run-Time Reconfiguration. Proceedings of the International Parallel Processing Symposium, IPPS 1998, 12th APPS and 9th SPDP, Orlando, Floride, États-Unis , 616-619.

DEHON, A. (1994). DPGA-coupled microprocessors: Commodity Ics for the early 21st century. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 31-39.

DEHON, A. (février 1996). DPGA Utilization and Applications. Proceedings of the 1996 International Symposium on Field Programmable Gate Array, Monterey, Californie, États-Unis.

DIESSEL, O., ELGINDY, H. (septembre 1997). Run-Time Compaction of FPGA Designs. Proceedings of the 7th International Workshop on Field Programmable Logic and Applications, Londres, Angleterre, 131.

DIESSEL, O., ELGINDY, H. (septembre 1998). Partial Rearrangements of Space-Shared FPGAs. Proceedings of the 8th International Workshop on Field Programmable Logic and Applications, 913.

DZUNG, T. H. (avril 1993). Searching Genetic Databases on Splash 2. Proceedings of the IEEE Workshop on FPGAs for Custom computing Machines, Napa, Californie, États-Unis, 185-191.

EBELING, C., McMURCHIE, L., HAUCK, S., BURNS, S. (1995). Placement and Routing Tools for the Tryptych FPGA. IEEE Transactions on Very Large Scale Integrated (VLSI) Systems, volume 3, numéro 4, 473-482.

EBELING, C., CRONQUIST, D. C., FRANKLIN, P., SECOSKY, J., BERG, S. G. (avril 1997). Mapping Applications to the RaPiD Configurable Architecture. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 106-115.

EGGERS, H., LYSAGHT, P., DICK, H., McGREGOR, G. (1996). Fast Reconfigurable Crossbar Switching in FPGAs. Field Programmable Logic - Smart Applications, New Paradigms and Compiler, Darmstadt, Allemagne, 297-306.

ELDREDGE, J. G., HUTCHINGS, B. L. (avril 1994). Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 180-188.

ELDREDGE, J. G., HUTCHINGS, B. L. (1994). RRANN: Run-Time Reconfiguration Artificial Neural Network. Proceedings of Custom Integrated Circuits Conference, 77-80.

FAURA, J., MORENO, J. M., AGUIRRE, M. A., Van DUONG, P., INSENSER, J. M. (septembre 1997). Multicontext Dynamic Reconfiguration and Real-time Probing on a Novel Mixed Signal Programmable Device with On-chip Microprocessor. Proceedings of the 7th International Workshop on Field Programmable Logic and Applications, Londres, Angleterre, 1.

FAWCETT, B. K. (septembre 1994). Using Teconfigurable FPGAs in Test Equipment Applications. Proceedings of the 1994 Wescon Conference, Anaheim, Californie, États-Unis, 562-567.

FOULK, P. W. (avril 1993). Data-folding in SRAM Configurable FPGAs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 163-171.

FRENCH, P.C., TAYLOR, R.W. (avril 1993). A Self-reconfiguring Processor. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 50-59.

GOKHALE, M., MARKS, A. (1995). Automatic Synthesis of Parallel Programs Targeted to Dynamically Reconfigurable Logic Arrays. Field Programmable Logic and Applications, Oxford, Angleterre, 399-408.

GUNTHER, B., MILNE, G., NARASIMHAN, L. (avril 1996). Assessing Document Relevance with Run-Time Reconfigurable Machines. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 10-17.

HADLEY, J. D., HUTCHINGS, B. L. (avril 1995). Design Methodologies for Partially Reconfigured Systems. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 78-84.

HASTIE, N., CLIFF, R. (mai 1990). The Implementation of Hardware Subroutines on Filed Programmable Gate Arrays. Proceedings of the Custom Integrated Circuits Conference.

HAUCK, S., FRY, T. W., HOSLER, M. M. (avril 1997). KAO, Jeffrey P., "The Chimaera Reconfigurable Functional Unit. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 87-96.

HAUCK, S., LI, Z., SCHWABE, E. (avril 1998). Configuration Compression for the Xilinx XC6200 FPGA. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 138-146.

HEEB, B., PFISTER, C. (septembre 1992). Chameleon: A Workstation of a Different Colour. Proceedings of the 2nd International Workshop on Field Programmable Logic and Applications.

HERON, J.-P., WOODS, R. F. (juillet 1998). Accelaring Run-Time Reconfiguration on Custom Computing Machines. Computer Arithmetic session SPIE conference, San Diego, États-Unis- , 19-24.

HUDSON, R. D., LEHN, D. I., ATHANAS, P. M. (avril 1998). A Run-Time Reconfigurable Engine for Image Interpolation. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 88-95.

HUTCHINGS, B. L., WIRTHLIN, M. J. (septembre 1995). Implementation Approaches for Reconfigurable Logic Applications. Proceedings of the 5th International Workshop on Field Programmable Logic and Applications, Oxford, Angleterre, 419-428.

HUTCHINGS, B., BELLOWS, P., HAWKINS, J., HEMMERT, S., NELSON, B., RYTTING, M. (avril 1999). A CAD Suite for High-Performance FPGA Design. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 12-24.

JAIN, S., BALAFRISHNAN, M., KUMAR, A., KUMAR, S. (janvier 1998). Speeding up Program Execution Using Reconfigurable Hardware and a Hardware Function Library. Proceedings of the IEEE International Conference on VLSI Design, Chennai, Inde, 400-405.

JOHNSON, B. W. (1989). Design and Analysis of Fault Tolerant Digital Systems. Addison Wesley, États-Unis.

KEAN, T. (septembre 1992). Using CAL to Accelerate Maze Routing of CAL Designs. Proceedings of the 2nd International Workshop on Field Programmable Logic, Vienne, Autriche.

KOZA, J. R., BENNETT, F. H. III, HUTCHINGS, J. L., BADE, S. L., KEANE, M. A., ANDRE, D. (1998). Evolving Computer Programs using Papidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming. Proceedings of the 1998 6th International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis , 209-219.

KUNG, S. (1988). VLSI Array Processors, Prentice Hall, États-Unis.

KWIAT, K. A., DEBANY, W. H. Jr., HARIRI, S. (mars 1996). Software Fault Tolerance Using Dynamically Reconfigurable FPGAs. Proceedings of the IEEE Great Lakes Symposium on VLSI, Ames, Iowa, États-Unis, 39-42.

LAZARUS, R. B., MEYER, F. M. (mai 1993). Realization of a Dynamically Reconfigurable Preprocessor. IEEE National Aerospace and Electrics Conference, 74-80.

LEMOINE, E., MERCERON, D. (avril 1995). Run Time Reconfiguration of FPGA for Scanning Genomic DataBases. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 90-98.

LI, J., CHENG, C.-K. (avril 1995). Routability Imporvement Using Dynamic Interconnect Architecture. IEEE Symposium on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 61-67.

LING, X.-P., AMANO, H. (avril 1993). WASMII: a Data Driven Computer on a Virtual Hardware. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 33-42.

LING, X.-P., AMANO, H. (septembre 1995). WASMII: An MPLD with Data-driven Control on a Virtual Hardware. Journal of Supercomputing, volume 9, numéro 3, 253-276.

LUDWIG, S. H.-M. (septembre 1996). The Design of a Coprocessor Board Using Xilinx's XC6200 FPGA - An Experience Report. Proceedins of the 6th International Workshop on Field Programmable Logic and Applications, Darmstadt, Allemagne, 77-86.

LUK, W., WU, T., PAGE, I. (avril 1994). Hardware-Software Codesign of Multidimensional Programs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 82-90.

LUK, W., SHIRAZI, N., CHEUNG, P. Y. K. (avril 1996). Modelling and Optimising Run-Time Reconfigurable Systems. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 167-176.

LUK, W., GUO, S., SHIRAZI, N., ZHUANG, N. (septembre 1996). A Framework for Developing Parametrised FPGA Librairies. 6th International Workshop on Field Programmable Logic: Smart Applications, New Paradigms and Compilers, Darmstadt, Allemagne, 24-33.

LUK, W., SHIRAZI, N., CHEUNG, P. Y. K. (avril 1997). Compilation Tools for Run-Time Reconfigurable Designs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 56-65.

LUK, W., LEE, T. K., RICE, J. R., CHEUNG, P. Y. K., SHIRAZI, N. (avril 1999). Reconfigurable Computing for Augmented Reality. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 136-145.

LYSAGHT, P. (1991). Dynamically Reconfigurable Logic in Undergraduate Projects. FPGAs: Proceedings of the 1991 International Workshop on Field Programmable Logic and Applications, Oxford, Royaume Uni, 424-436.

LYSAGHT, P., DUNLOP, J. (septembre 1993). Dynamic Reconfiguration of FPGAs. More FPGAs, Proceedings of the 1993 International Workshop on Field Programmable Logic and Applications, Oxford, Royaume Uni, 82-94.

LYSAGHT, P., DICK, H. P. (septembre 1994). Implementation of Adaptative Signal Processing Architectures Based on Dynamically Reconfigurable FPGAs. Proceedings of EUSIPCO-94, European Signal Processing Conference, Edinburgh, Écosse, 1871-1874.

LYSAGHT, P., McCONNELL, D., DICK, H. (septembre 1994). Design Experience with Fine-Grained FPGAs. Proceedings of the 4th International Workshop on Field Programmable Logic and Applications, Prague, République Tchèque, 298-302.

LYSAGHT, P., STOCKWOOD, J., LAW, J., GIRMA, D. (septembre 1994). Artificial Neural Network Implementation on a Fine-Grained FPGA. Field Programmable Logic: Architecture Synthesis ans Applications, Prague, République Tchèque, 421-431.

LYSAGHT, P., DICK, H., McGREGOR, G., McCONNELL, D., STOCKWOOD, J. (septembre 1995). Prototyping Environment for Dynamically Reconfigurable Logic. Proceedings of the 5th International Workshop on Field Programmable Logic and Applications, Oxford, Angleterre, 409-418.

LYSAGHT, P., McGREGOR, G., STOCKWOOD, J. (février 1996). Configuration Controller Synthesis for Dynamically Reconfigurable Systems. Proceedings of the 1996 IEE Colloquium Hardware-Software Cosynthesis for Reconfigurable Systems, Londres, Angleterre.

LYSAGHT, P., STOCKWOOD, J. (septembre 1996). A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays. IEEE Transactions on very large scale integration (VLSI) systems, volume 4, numéro 3, 381-390.

LYSAGHT, P. (septembre 1997).Towards an Expert System for a priori Estimation of Reconfiguration Latency in Dynamically Reconfigurable Logic. Proceedings of the 7th International Workshop on Field Programmable Logic and Applications, Londres, Angleterre, 183-192.

MAKI, G., WHITAKER, S., GANESH, G. (septembre 1991). Reconfigurable Data Path Processor. Fourth annual IEEE International ASIC Conference and Exhibit, P18-4.1 - P18-4.

McGREGOR, G., LYSAGHT, P. (septembre 1997). Extending Dynamic Circuit Switching to Meet the Challenges of New FPGA Architectures. Proceedings of the 7th International Workshop on Field Programmable Logic and Applications, Londres, Angleterre, 31-40.

McGREGOR, G., ROBINSON, D., LYSAGHT, P. (septembre 1998). Hardware/Software Co-Design Environment for Reconfigurable Logic Systems. Proceedings of the 8th International Workshop on Field Programmable Logic and Applications, 258.

McKAY, N., SINGH, S. (avril 1999). Debugging Techiques for Dynamically Reconfigurable Hardware. IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, Californie, États-Unis , 114-122.

MORENO, J. M., MADRENAS, J., FAURA, J., CANTO, E., CABESTANY, J., INSENSER, J. M. (septembre 1998). Feasible Evolutionary and Self-Reparing Hardware by Means of the Dynamic Reconfiguration of the FIPSOC Devices. Proceedings of the 8th International Workshop on Field Programmable Logic and Applications, 345.

PARK, S. R., BURLESON, W. (1998). Reconfiguration for Power Saving in Real-time Motion Estimation. IEEE International Conference on Acoustics, speech and signal processing ICASSP, Seatle, Washington, États-Unis , 3037-3040.

RABEL, C. E., SAWAN, M. (mai 1999). PARC: A New Pyramidal FPGA Architecture Based on a RISC Processor. IEEE ISCAS - International Symposium on Circuit and Systems, Orlando, Floride, États-Unis.

ROBINSON, D., LYSAGHT, P., McGREGOR, G., DICK, H. (septembre 1997). Performance Evaluation of a Full Speed PCI Initiator and Target Subsystem Using FPGAs. Proceedings of the 7th International Workshop on Field Programmable Logic and Applications, Londres, Angleterre, 41.

ROBINSON, D., McGREGOR, G., LYSAGHT, P. (août 1998). New CAD Framework Extends Simulation of Dynamically Reconfigurable Logic. Field Programmable Logic and Applications 8th International Workshop, Tallinn, Estonie, 1-8.

ROSS, D., VELLACOTT, O., TURNER, M. (septembre 1993). An FPGA-based Hardware Accelerator for Image Processing. More FPGAs, Proceedings of the 1993 International Workshop on Field Programmable Logic and Applications, Oxford, Royaume Uni, 299-306.

SCHMIT, H. (avril 1997). Incremental Reconfiguration for Pipelined Applications. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 47-55.

SCHONER, B., JONES, C., VILLASENOR, J. (avril 1995). Issues in Wireless Video Coding using Run-time-reconfigurable FPGAs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 85-89.

SEZER, S., HERON, J., WOODS, R., TURNER, R., MARSHALL, A. (1998). Fast Partial Reconfiguration for FCCMs. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 318-319.

SHIBATA, Y., LING, X.-P., AMANO, H. (septembre 1996). Emulation System of the WASMII: a Data Driven Computer on a Virtual Hardware. Proceedings of the 6th International workshop on Field Programmable Logic and Applications, Darmstadt, Allemagne, 55.

SHIBATA, Y., MIYARAKI, H., LING, X.-P., AMANO, H. (octobre 1997). Towards the Realistic "Virtual Hardware". Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems, Maui, Hawaii, États-Unis, 50-55.

SHIRAZI, N., LUK, W., CHEUNG, P. Y. K. (avril 1998). Automating Production of Run-Time Reconfigurable Designs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 147-156.

SINGH, S., BELLEC, P. (avril 1994). Virtual Hardware for Graphics Applications Using FPGAs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 49-58.

SINGH, S., HOGG, J., McAULEY, D. (avril 1996). Expressing Dynamic Reconfiguration by Partial Evaluation. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 188-194.

STOCKWOOD, J., LYSAGHT, P. (septembre 1995). Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays. Proceedings of the Annual IEEE International ASIC Conference and Exhibit, Austin, Texas, États-Unis, 167-170.

TAU, E., ESLICK, I., CHEN, D., BROWN, J., DEHON, A. (mai 1995). A First Generation DPGA Implementation. Proceedings of the Third Canadian Workshop on Field-Programmable Devices, 138-143.

TRIMBERGER, S., CARBERRY, D., JOHNSON, A., WONG, J. (avril 1997). A Time-Multiplexed FPGA. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 22-28.

VAN DAALEN, M., JEAVONS, P., SHAWE-TAYLOR, J. (avril 1993). A Stochastic Neural Architecture that Exploits Dynamically Reconfigurable FPGAs. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 202-211.

VASILKO, M., AIT-BOUDAOUD, D. (septembre 1996). Architectural Synthesis Techniques for Dynamically Reconfigurable Logic. Field Programmable Logic and Applications, Darmstadt, Allemagne, 290-296.

VASILKO, M., LONG, D. (juin 1998). RIFLE-62: A Flexible Environment for Prototyping Dynamically Reconfigurable Systems. Proceedings of the International Workshop on Rapid System Prototyping , Leuven, Belgique, 130-135.

VASILKO, M., CABANIS, D. (avril 1999). Improving Simulation Accurancy in Design Methodologies for Dynamically Reconfigurable Logic Systems. IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, Californie, États-Unis , 123-133.

VILAASENOR, J., SCHONER, B., CHIA, K.-N., ZAPATA, C. (avril 1996). Configurable Computing Solutions for Automatic Target Recognition. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 70-79.

VILLASENOR, J., MANGIONE-SMITH, W. H. (june 1997). Configurable Computing. Scientific American, 66-71.

VILLASENOR, J., HUTCHINGS, B. (septembre 1998). The Flexibility of Configurable Computing. IEEE Signal Processing Magazine, 67-84.

VUILLEMIN, J. E., BERTIN, P., RONCIN, D., SHAND, M., TOUATI, H. H., BOUCARD, P. (mai 1996). Programmable Active Memories: Reconfigurable Systems Come of Age. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 4, numéro 1, 56-69.

WIRTHLIN, M. J., HUTCHINGS, B. L. (avril 1995). A Dynamic Instruction Set Computer. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis , 99-107.

WIRTHLIN, M. J., HUTCHINGS, B. L. (1996). Sequencing Run-Time Reconfigured Hardware with Software. Proceedings of the 1996 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis , 122-128.

WIRTHLIN, M. J., HUTCHINGS, B. L. (1997). Improving Functional Density Through Run-Time Constant Propagation. Proceedings of the 1997 5th International Symposium on Field Programmable Gate Arrays, Monterey, Californie, États-Unis , 86-92.

WIRTHLIN, M. J., HUTCHINGS, B. L. (juin 1998). Improving Functional Density Using Run-Time Circuit Reconfiguration. IEEE transactions on Very Large Scale Integration (VLSI) systems, volume 6, numéro 2, 247-256.

WOODS, R., LUDWIG, S., HERON, J., TRAINOR, D., GEHRING, S. (avril 1997). FPGA Synthesis on the XC6200 using IRIS and TRIANUS/HADES (or from Heaven to Hell and back again). Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Napa, Californie, États-Unis, 155-164.

Xilinx, Databook, 1998, San Jose, Californie, États-Unis

ZHANG, X.-J., NG, K.-W., YOUNG, G. H., (septembre 1997). High-Level Synthesis Using Genetic Algorithms for Dynamically Reconfigurable FPGAs. Conference Proceedings of the EUROMICRO New Frontiers of Information Technology, Budapest, Hongrie, 234-243.