Titre: Title:	Méthodologie de codesign pour les systèmes sur puce programmable
Auteur: Author:	Loïc Pierron
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Pierron, L. (2005). Méthodologie de codesign pour les systèmes sur puce programmable [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7665/

UNIVERSITÉ DE MONTRÉAL

MÉTHODOLOGIE DE CODESIGN POUR LES SYSTÈMES SUR PUCE PROGRAMMABLE

LOÏC PIERRON
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
DÉCEMBRE 2005

Your file Votre référence
ISBN: 978-0-494-16834-9
Our file Notre référence
ISBN: 978-0-494-16834-9

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MÉTHODOLOGIE DE CODESIGN POUR LES SYSTÈMES SUR PUCE PROGRAMMABLE

présenté par: PIERRON Loïc

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

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

M. ABOULHAMID El Mostapha, Ph.D., président

M. BOIS Guy, Ph.D., membre et directeur de recherche

Mme. NICOLESCU Gabriela, Doct., membre

To us.

REMERCIEMENTS

Tout d'abord, je remercie mon directeur de recherche, Guy Bois, pour son aide financière m'ayant permis de commencer ma maîtrise dans de bonnes conditions.

Je remercie ensuite énormément les personnes qui m'ont aidé en fin de parcours : d'une part mes parents, et surtout ma maman, Nadine, pour leur soutien financier ; d'autre part mon frère d'armes, Quentin Bleton, qui m'a gracieusement accueilli chez lui pendant presque six mois, et sur lequel je peux vraiment compter.

Pour l'efficacité avec laquelle ils ont régulièrement répondu à mes besoins et réglé mes problèmes, je remercie les administrateurs systèmes du groupe de recherche, Réjean Lepage et Alexander Vesey. Je remercie également Marc Minato, de la compagnie Mentor Graphics, pour son support technique irréprochable à un moment crucial de mon travail.

RÉSUMÉ

Depuis le début de la microélectronique, les performances des puces disponibles n'ont cessé de s'améliorer, mais c'est actuellement dans le domaine des puces à architecture reprogrammable que les progrès sont les plus importants, ces progrès dépassant même les prévisions de la loi de Moore au cours des dix dernières années. Le succès croissant des FPGA s'explique également par l'apparition encore récente des FPGA plateformes, des FPGA sur lesquels, en plus des blocs logiques programmables et du réseau d'interconnexions programmable, ont été ajoutées en matériel des propriétés intellectuelles comme un microprocesseur.

Compte tenu de la taille et de la complexité toujours croissantes des systèmes sur puces, l'utilisation de méthodologies et d'outils appropriés est primordiale si l'on souhaite pouvoir tirer profit au maximum des possibilités offertes par ces nouvelles puces. La réutilisation de la propriété intellectuelle, par exemple, fait déjà partie intégrante des flots de conception pour FPGA. Dans ce travail, vous avons exploré une autre piste, celle de la cosimulation, et étudié comment l'utilisation d'un outil de covérification comme Mentor Seamless CVE peut bénéficier au flot de conception classique pour Xilinx Virtex-II Pro.

Nous avons commencé par créer un modèle cosimulable d'un design de base. À partir de cette expérience, nous avons établi une méthode de transformation d'un design quelconque obtenu avec les outils Xilinx vers un modèle cosimulable dans Seamless. Nous avons fait rouler un système d'exploitation temps réel avec ce modèle, puis à partir de cette plateforme de référence, nous avons défini une méthodologie de codesign pour systèmes sur puce programmable, adaptée au flot de conception Xilinx et utilisant la cosimulation. Nous sommes capables de cosimuler des designs possédant des modules matériels modélisés à différents niveaux d'abs-

traction et communiquant avec le processeur à différents niveaux d'abstraction également. Notre méthodologie permet donc de concevoir des modules matériels par raffinement progressif et de valider leur fonctionnement à chaque niveau d'abstraction.

Afin de démontrer le fonctionnement de la méthodologie mise au point, nous l'avons appliquée à un exemple de détection de contours, et nous avons mesuré et analysé les résultats obtenus. En conclusion, l'intégration de la cosimulation dans le flot de conception FPGA permet d'accélérer le cycle de conception d'un design, et en particulier de faciliter grandement la phase d'exploration architecturale.

ABSTRACT

Since the beginning of microelectronics, chip performance has constantly been improved, but nowadays the most significant progress is being made in the field of reprogrammable architecture chips, so that it even surpassed Moore's law's predictions in the last ten years. The increasing success of FPGAs also comes from the still recent platform FPGAs, FPGAs that don't only feature reprogrammable logic blocks in a reprogrammable interconnection network, but also integrate hardwired IP cores such as microprocessors.

Because of the exploding complexity of system-on-chip design, it is extremely important to use adequate methodologies and tools in order to harness all the possibilities offered by these new chips. IP reuse, for example, is already a part of current FPGA design flows. In this work, we explored the idea of using co-simulation, and studied how the use of a co-verification tool like Mentor Seamless CVE can improve the classical design flow for Xilinx Virtex-II Pro.

From the experience of creating a first co-simulation model of a reference design, we established a way to transform any design obtained with Xilinx' tools into a co-simulation model for Seamless. We had a real-time operating system run with this model, and we defined a co-design methodology for system-on-programmable-chip design that is adapted to the classical Xilinx design flow and uses co-simulation. We are able to co-simulate designs integrating hardware modules modelled at different abstraction levels and communicating with the processor at different abstraction levels as well. Our methodology therefore allows us to design hardware modules through progressive refining and validate them at each abstraction level.

To demonstrate how our methodology works, we applied it to accelerate an edges detection algorithm, and we measured and analyzed the results. To conclude, inte-

grating the use of co-simulation into the classical FPGA design flow can accelerate the design cycle, in particular by facilitating the architectural exploration phase.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AHB	Advanced High Performance
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BRAM	BlockRAM
CAM	Content-Addressable Memory
CPLD	Complex Programmable Logic Device
CVE	Co-Verification Environment
DCR	Device Control Register
DDR	Double Data Rate
DPRAM	Dual Port RAM
EDK	Embedded Development Kit
ESB	Embedded System Block
FIFO	First In First Out
FIT	Fixed Interval Timer
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IP	Intellectual Property
IPIC	IP InterConnect
IPIF	IP InterFace
ISE	Integrated Software Environment
ISS	Instruction Set Simulator
MGT	Multi-Gigabit Transceiver
MMU	Memory Management Unit

OPB	On-chip Peripheral Bus
PCI	Peripheral Component Interconnect
PIT	Programmable Interval Timer
PLA	Programmable Logic Array
PLB	Processor Local Bus
PMC	PCI Mezzanine Card
PSP	Processor Support Package
RAM	Random Access Memory
ROM	Read Only Memory
RTOS	Real Time Operation System
SDRAM	Synchronous Dynamic RAM
SoC	System-on-Chip
SPP	Seamless Performance Profiler
SRAM	Static RAM
UART	Universal Asynchronous Receiver/Transmitter
VHDL	Very high speed integrated circuit HDL
XPS	Xilinx Platform Studio

INTRODUCTION

Aujourd'hui, nous sommes entourés d'appareils électroniques de plus en plus variés, de plus en plus petits, de plus en plus puissants. Prenons l'exemple du téléphone cellulaire. Cette petite merveille de technologie qui tient dans la main peut, entre autres choses, prendre des photos, lire des films, envoyer et recevoir des courriels, jouer de la musique, et, chose à laquelle peu de personnes font encore attention, nous permettre de parler en temps réel à n'importe qui sur la planète.

Le téléphone cellulaire est un excellent exemple de système embarqué, un système informatique - matériel et logiciel, dédié à l'exécution de tâches spécifiques. Il contient un processeur qui exécute du code (la partie logicielle), et il contient du matériel spécialisé pour certains traitements, le tout relié par un système d'interconnexions permettant aux différents composants de communiquer entre eux.

Mais avant d'en arriver là, il a non seulement fallu que les chercheurs et les ingénieurs inventent de nouveaux composants, mais aussi qu'ils réduisent énormément leur taille. C'est justement un des principaux accomplissements de la microélectronique, qui est le cœur de l'électronique d'aujourd'hui. Pour bien comprendre comment on en est arrivé aux technologies actuelles, il est intéressant de revenir un peu en arrière dans notre histoire, de disons deux siècles.

Historique de l'électronique et de la microélectronique

Le dix-neuvième siècle a été celui de la révolution industrielle. Parallèlement au triomphe de la vapeur et de la force thermodynamique, il a également marqué une période de grandes découvertes dans le domaine de l'électromagnétisme. Alors qu'au début du siècle, on était tout juste capable de construire la première pile, la

fin du siècle voyait l'apparition de la machine à rayons X. Grâce aux applications de l'électromagnétisme, le dix-neuvième siècle a également été celui des premières communications, avec des appareils comme le télégraphe puis le téléphone.

Au cours du siècle dernier, tout s'est accéléré. En effet, dès le début du vingtième siècle, toutes sortes d'appareils électriques ont fait leur apparition dans les foyers et ont progressivement changé la vie de tout un chacun. Du côté des communications, on apprend à se passer de médium de transmission métallique et c'est la naissance des communications sans fil, dont les applications mènent à la radio et à la télévision. Mais jusqu'à la moitié du vingtième siècle, ces premiers appareils de communication sans fil sont très encombrants et peu fiables.

Une invention va changer tout ça et lancer la révolution électronique : celle du transistor, en 1947. Le transistor est utilisé en remplacement du tube électronique, jusqu'alors notamment responsable d'amplifier le courant dans les circuits, et est non seulement plus petit, mais aussi plus fiable et moins coûteux. On se met donc à fabriquer des postes de radio portables, moins chers, et offrant à la fois AM et FM. Ces mêmes postes de radio "à transistors" ont d'ailleurs longtemps été incorrectement appelés transistors.

Les premiers transistors sont très vite remplacés par des transistors faits à partir de matériaux semiconducteurs. Tout d'abord de germanium, puis de silicium, qui est encore aujourd'hui principalement utilisé, plus de cinquante ans après ses débuts en 1954. Progressivement, on met de plus en plus de transistors sur une seule et même tranche de silicium, avant de la découper pour produire les composants discrets.

Puis vient l'idée de fabriquer des puces de silicium contenant non seulement des transistors, mais aussi d'autres composants électroniques de base comme des résistances et des condensateurs, permettant ainsi de mettre un circuit électronique complet sur une puce. En 1958, seulement une dizaine d'années après l'invention

du premier transistor, c'est la naissance des circuits intégrés.

De même que le transistor a révolutionné le domaine des communications, il a aussi, par l'intermédiaire des circuits intégrés, révolutionné celui des ordinateurs. Alors que les ordinateurs du milieu du vingtième siècle occupaient des salles entières et étaient réservés à des groupes d'utilisateurs privilégiés, l'apparition du microprocesseur au début des années 1970 permit dès la fin de la décennie de construire des ordinateurs personnels tenant sur un bureau [Museum (2005)]. Depuis, la taille des transistors n'a cessé de diminuer, et on s'est mis à parler de microélectronique.

Les mécanismes de l'évolution en électronique

Avant de tourner cette page historique, essayons de voir comment se fait l'évolution dans le domaine de l'électronique. Prenons le cas de la télévision. La télévision a ceci d'intéressant qu'elle a relativement peu évolué - la couleur existait déjà - depuis le début de sa démocratisation, au milieu du siècle dernier. En effet, en dehors de quelques innovations récentes et encore peu répandues comme la télévision haute définition, la télévision a gardé le même principe de fonctionnement. Cependant, au fil des années, les téléviseurs se sont faits de plus en plus petits, de moins en moins chers, de plus en plus performants et de plus en plus perfectionnés ; la télécommande est devenue son inséparable compagne, la qualité de l'image ainsi que celle du son se sont améliorées.

Derrière ces améliorations, il y a plusieurs phénomènes. Tout d'abord, il y a la miniaturisation des composants électroniques. Cette miniaturisation a plusieurs conséquences bénéfiques : bien évidemment, la taille des circuits électroniques se trouve diminuée en conséquence, et on peut donc construire des téléviseurs plus petits et plus légers ; si on souhaite en revanche construire des téléviseurs de même taille, on peut alors utiliser l'espace gagné pour ajouter de nouvelles fonctionnalités

au téléviseur. Il y a aussi l'amélioration des composants électroniques existants, ce qui permet d'améliorer la performance des circuits, et il y a encore l'apparition de nouveaux composants, qui permet quant à elle de créer de nouveaux types de circuits.

Cette évolution est généralisable à la plupart des domaines de l'électronique, et notamment à celui qui nous intéresse plus particulièrement, celui de la microélectronique. Au niveau de la miniaturisation, les progrès réalisés au cours des années sont considérables. On parle aujourd'hui de transistors dont les dimensions peuvent être aussi petites que 90 nanomètres (et bientôt 65 nm), et on fabrique des puces de grande consommation destinées aux ordinateurs contenant plus de cent millions de transistors, là où le premier microprocesseur, fabriqué par Intel en 1971, contenait environ 2300 transistors.

Il y a eu évolution au niveau des types de composants, avec par exemple la découverte du transistor à effet de champ, mais aussi au niveau des technologies de fabrication, avec par exemple la technologie CMOS. On a également vu l'apparition de nouvelles classes de circuits ou de puces. Au début des années 1970, peu après l'invention du premier microprocesseur, les puces à architecture programmable ont fait leur apparition. Ces circuits (PLD puis CPLD, FPGA), initialement utilisés pour des fonctions combinatoires simples, sont désormais très versatiles et peuvent mélanger des éléments aussi divers qu'une simple porte logique ou un microprocesseur complet.

Les défis de la microélectronique

Les défis posés par la microélectronique sont multiples et complexes. D'une manière évidente, il y a une grande différence d'accessibilité entre une puce et un circuit électronique classique. En effet, Monsieur Tout-le-Monde peut très facile-

ment, avec très peu d'équipement spécialisé (en gros, une insoleuse et quelques produits chimiques), fabriquer ses propres circuits imprimés. Ensuite, il peut acheter des composants dans le magasin d'électronique le plus proche, se munir de son fer à souder, et le reste n'est qu'une question d'huile de coude. Même s'il s'agit de composants CMS (Composants Montés en Surface, en anglais SMC pour Surface Mounted Components), chaque composant est macroscopique et aisément manipulable.

En revanche, pour fabriquer une puce contenant des millions de transistors concentrés sur moins d'un centimètre carré, il faut adopter une approche totalement différente. Même avec une excellente vue et une incroyable dextérité, il est humainement impossible (aujourd'hui du moins) de graver une tranche de silicium manuellement. En microélectronique, dès qu'on veut produire, il faut donc obligatoirement utiliser des machines. Pour que la microélectronique continue de progresser, il a donc fallu sans cesse améliorer les procédés de fabrication, et donc les machines dont ils dépendent.

En fait, depuis longtemps, on utilise également des machines pour aider à la conception des puces. Le microordinateur, qui est l'un des principaux produits de la microélectronique, a très vite été utilisé pour permettre à la microélectronique d'aller encore plus loin. Même si ça semble ironique, on ne pourrait pas améliorer les ordinateurs sans les ordinateurs eux-mêmes ! L'utilisation de l'ordinateur comme aide à la conception n'est pas propre à la microélectronique et, d'une manière générale, on appelle ça la conception assistée par ordinateur (CAO, en anglais CAD pour Computer-Aided Design).

Pour progresser, la microélectronique a donc bénéficié de la création de nouvelles méthodologies de conception et de l'utilisation d'outils informatiques spécialisés. Ces outils de conception électronique automatisée (en anglais EDA tools, pour

Electrical Design Automation tools) constituent un domaine de recherche à part entière de la microélectronique. Méthodologies et outils vont souvent de paire, mais on peut même considérer que tous les aspects de la microélectronique sont interdépendants. En effet, si une nouvelle technologie est mise au point mais qu'il n'existe aucun outil approprié, cette technologie reste alors inexploitable pour les concepteurs de puces.

Les outils développés et les méthodologies mises au point sont dépendants de la technologie visée. Par exemple, le flot de conception sera sensiblement différent selon qu'on veuille produire des ASIC (Application Specific Integrated Circuit) ou des FPGA (Field Programmable Gate Array). Cependant, certaines étapes sont communes à toutes les méthodologies, quelle que soit la technologie cible. On peut même, dépendamment de l'application, avoir des étapes, dans le flot de conception, qui sont complètement indépendantes de la technologie visée.

Dans ce mémoire, nous nous penchons sur l'étude d'une méthodologie de codesign sur FPGA plateforme (de l'anglais platform FPGA). Elle s'articule autour de l'utilisation de deux outils en particulier, et se propose de réaliser une intégration, un mélange des flots de conception proposés par chacun d'eux, afin d'accélérer le processus de conception sur ce type de puces.

Problématique

Puisque le nombre de transistors qu'il est possible d'intégrer augmente régulièrement, la taille et la complexité des systèmes embarqués ne cessent d'augmenter. Ce phénomène touche notamment les systèmes sur puce programmable, qui en plus jouissent d'un succès croissant. Afin de concevoir efficacement des systèmes, il est devenu nécessaire d'utiliser des outils d'aide à la conception électronique, mais les outils seuls ne peuvent pas tout faire. Aussi est-il important de définir des métho-

dologies permettant non seulement de tirer profit au maximum des outils, mais aussi de définir une approche systématique à la conception de systèmes.

Nous nous intéressons au développement de systèmes sur puce programmable Virtex-II Pro en particulier parce que notre laboratoire allait être équipé de cartes de développement et de prototypage équipées de ce type de puce. Naturellement, la compagnie Xilinx fournit une suite d'outils pour la conception de systèmes sur ces FPGA, et ces outils feront partie intégrante de la méthodologie mise au point.

Un autre aspect que nous voulons explorer est celui de la cosimulation. La compagnie Mentor venait d'annoncer le support du PowerPC 405 D5, le processeur contenu dans les Virtex-II Pro, dans son outil Seamless CVE. L'utilisation de la cosimulation permettrait par ailleurs de créer un design de référence et des applications pour le Virtex-II Pro avant de recevoir les cartes de développement. Seamless venant évidemment lui aussi avec sa propre méthodologie de conception, nous allons regarder comment il est possible de combiner les outils et les méthodologies afin de créer une nouvelle méthodologie de conception de systèmes sur puce programmable, le but étant d'améliorer la vitesse ou la facilité de conception.

Objectifs

Le premier objectif lors de ce travail était simplement de se familiariser avec les outils offerts par Xilinx, et d'évaluer le support du Virtex-II Pro dans Seamless en tentant de créer un design de référence.

Lorsque le laboratoire a reçu les cartes de développement, le principal objectif est devenu d'obtenir un modèle de cosimulation du design de référence fourni avec les cartes Amirix dans Seamless.

Enfin, l'objectif final de ce travail est de mettre au point une méthodologie de

codesign, adaptée à la conception de systèmes sur puce programmable, pouvant servir de complément à l'utilisation des cartes.

Méthodologie

Nous avons commencé par utiliser les outils chacun de leur côté, afin de comprendre leur fonctionnement, et de trouver comment il était possible de les combiner dans un flot de conception unique, permettant entre autres d'obtenir facilement un modèle de cosimulation à partir d'une plateforme conçue avec les outils de Xilinx.

Nous avons ensuite commencé à utiliser ces outils de manière combinée afin de créer des plateformes très simples, avant d'entreprendre la mise au point du modèle pour le design de référence des cartes de prototypage. Ceci nous a permis de créer une technique largement automatisée pour obtenir le modèle cosimulable dans Seamless d'un design conçu avec l'outil XPS (Xilinx Platform Studio).

Enfin, pour accélérer et faciliter la création de modules matériels, nous avons exploré toutes les diverses possibilités offertes par Seamless et nous nous sommes efforcés de les inclure dans une méthodologie de codesign adaptée à la conception de systèmes sur puce reprogrammable.

Originalité et contributions

Pour ce qui est de l'originalité, lorsque ce projet à été commencé, l'idée d'associer XPS et Seamless dans un flot de conception unique, et surtout celle d'automatiser l'obtention de modèles de cosimulation Seamless à partir d'un sytème conçu avec XPS étaient toutes deux nouvelles. Depuis, les deux compagnies, ont ajouté dans leurs outils respectifs plusieurs assistants dont l'utilisation combinée permet d'au-

tomatiser grandement le passage d'XPS vers Seamless. Il reste que nous avons fait la même chose avant eux.

Du côté des contributions, on notera surtout la création d'une méthodologie de création de modules matériels par raffinement progressif, rendue possible grâce à l'utilisation conjointe de caractéristiques propres à chacun des outils. Cette méthodologie est utile de la phase d'exploration architecturale à la phase d'implémentation des modules. Il y a aussi le fait que la plateforme finale obtenue pour l'exemple d'application présenté intègre toutes les caractéristiques possibles de Seamless. Cette plateforme représente donc non seulement un outil de recherche et développement, mais aussi un outil éducatif sur Seamless et le codesign en général.

Plan du mémoire

Entre sa présente introduction et sa conclusion, ce mémoire est constitué de quatre chapitres. Le chapitre 1 fait l'état de l'art dans le domaine du codesign et des FPGA. On y aborde tout d'abord des sujets vastes avant de préciser les choses et d'arriver à ce qui nous intéresse plus particulièrement : les FPGA plateformes et la cosimulation. Le chapitre 2 présente la carte de développement et le design de référence ciblés. Ceci définit le cadre dans lequel le travail a été réalisé, et explique certains des choix qui ont été faits par la suite. Le chapitre 3 définit et explique, étape par étape, la méthodologie de conception de systèmes sur puce programmable mise au point dans ce travail. Le chapitre 4 illustre cette méthodologie à l'aide d'un exemple de détecteur de contours utilisant les masques de Sobel, et présente et discute les résultats obtenus.

CHAPITRE 1

ÉTAT DE L'ART

La microélectronique est un domaine de recherche très vaste, et il est donc important de situer un peu plus précisément le travail dont il est question dans ce mémoire. Nous nous intéressons ici à une méthodologie de codesign de systèmes sur puces embarqués.

1.1 Les SoC

Les systèmes sur puce (en anglais SoC pour System on Chip) sont en quelque sorte la seconde étape d'intégration après les circuits intégrés. Là où les circuits intégrés ont révolutionné l'électronique en rassemblant plusieurs composants sur une seule puce (microprocesseur, mémoire, puce de décodage video, etc.), le système sur puce va plus loin, en combinant ce qui était autrefois sur plusieurs puces sur une seule et même puce. On a désormais un système complet (au minimum un processeur, de la mémoire, et un réseau d'interconnexions) sur une seule puce.

Les SoC sont des systèmes mixtes combinant logiciel et matériel, et ils sont très adaptés à la conception de systèmes embarqués. Ils permettent donc de tirer profit à la fois des avantages du logiciel (versatilité, prix...) et de ceux du matériel (performance...) afin de trouver le meilleur compromis pour l'application désirée.

Un avantage important des SoC, en outre celui de la possibilité d'intégration, est celui de la vitesse. Puisque tout se fait à l'intérieur d'une puce, les communications entre les différents modules du système, par exemple à travers un bus sur puce,

sont beaucoup plus rapides qu'elles le seraient entre différentes puces reliées par un bus.

Les SoC apportent également leur lot d'inconvénients. Comme ils contiennent une faune diversifiée de modules pouvant réaliser des fonctions très différentes, ils peuvent être extrêmement complexes à concevoir, surtout avec les progrès croissants au niveau des technologies de fabrication. Alors que l'écart de productivité dû à la différence entre le nombre de transistors qu'il est possible d'intégrer sur une puce et le nombre de transistors que peuvent gérer les concepteurs a été grandement réduit aux cours des précédentes décennies grâce au développement des outils de conception électronique automatisée, on fait désormais face à un nouvel écart de productivité cette fois entraîné par la différence entre la complexité des technologies des procédés de fabrication et les capacités de conception [Wakabayashi et Okamoto (2004)].

Pour pallier les problèmes rencontrés dans la conception de SoC, il est nécessaire de développer de nouveaux outils et de nouvelles façons de concevoir, des méthodologies.

1.2 Le codesign

Une méthodologie de codesign, ou de conception conjointe logicielle/matérielle, est une méthodologie de conception qui combine les flots de conception matériel et logiciel à l'intérieur d'un flot unique. Bien évidemment, le produit final aura des parties matérielles et des parties logicielles, mais une méthodologie de codesign prévoit et permet de gérer cette dualité dans la nature du système tout au long du processus de conception.

La figure 1.1 présente les étapes générales d'une méthodologie de codesign.

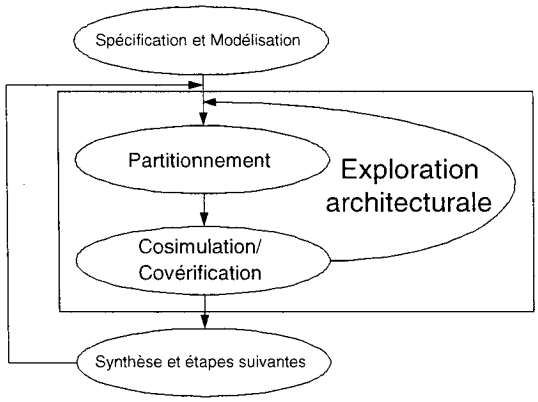

Figure 1.1 Les étapes du codesign

Nous avons ici divisé la méthodologie de codesign en cinq étapes (ou quatre plus une, l'exploration architecturale étant plutôt un ensemble de sous-étapes) :

- Spécification et modélisation : il s'agit de traduire le cahier des charges dans un langage (ou une représentation) plus approprié pour la suite du flot de conception. Il peut en résulter la mise au point d'un modèle exécutable écrit à haut niveau d'abstraction, mais encore indépendant des futurs choix architecturaux.

- Partitionnement : on choisit comment organiser le système d'un point de vue architectural, et en particulier on décide quelles parties seront exécutées en logiciel et lesquelles seront implémentées sous forme de bloc matériel.

- Cosimulation ou covérification : le design ayant été partitionné en un système mixte logiciel-matériel, on le simule grâce à un outil permettant la simulation hétérogène, afin de vérifier le système et valider le choix de partitionnement dont il découle.

- Exploration architecturale : l'exploration architecturale vise, par une succession d'itérations au cours de chacune desquelles on réalise un partitionnement et la cosimulation correspondante, à trouver dans un premier temps des solutions res-

pectant les contraintes de la spécification, puis dans un deuxième temps à identifier la solution qui offre le meilleur compromis performance/prix/versatilité.

- Synthèse et étapes suivantes : une fois l'exploration architecturale terminée, et le choix de partitionnement arrêté, on peut passer à la suite du flot de conception, qui sera alors le même que dans une méthodologie classique, i.e. synthèse, placement, routage, etc. Il se peut néanmoins qu'apparaisse ici une violation de contrainte, ce qui entraîne alors des coûts beacoup plus importants que lorsque ça arrive à l'étape précédente.

Dans une méthodologie de conception classique (par opposition à une méthodologie de codesign), le logiciel et le matériel sont développés séparément. Ceci pose quelques problèmes sérieux : premièrement, avant de pouvoir tester le logiciel, il est nécessaire de disposer d'un prototype fonctionnel du matériel. L'avancement côté logiciel est donc bloqué jusqu'à ce que le développement côté matériel atteigne une phase presque terminale. Deuxièmement, puisque le logiciel et le matériel sont développés chacun de leur côté, leur intégration est une étape très problématique.

Grâce à la cosimulation/covérification, qui constitue une étape cruciale dans la méthodologie de codesign, il est possible de réduire la durée du développement d'un produit. Dès qu'on a une version simulable du matériel, on peut réaliser une cosimulation logicielle/matérielle, et ainsi commencer à tester l'exécution du logiciel sur le matériel. Plus l'outil de cosimulation utilisé supporte des modèles matériels décrits à haut niveau d'abstraction, plus tôt on peut commencer à tester le logiciel et donc plus le temps d'accès au marché peut être diminué. La figure 1.2 illustre comment l'utilisation de la cosimulation permet d'accélérer le cycle de développement d'un produit.

Le codesign présente plusieurs avantages intéressants (par rapport aux méthodolo-

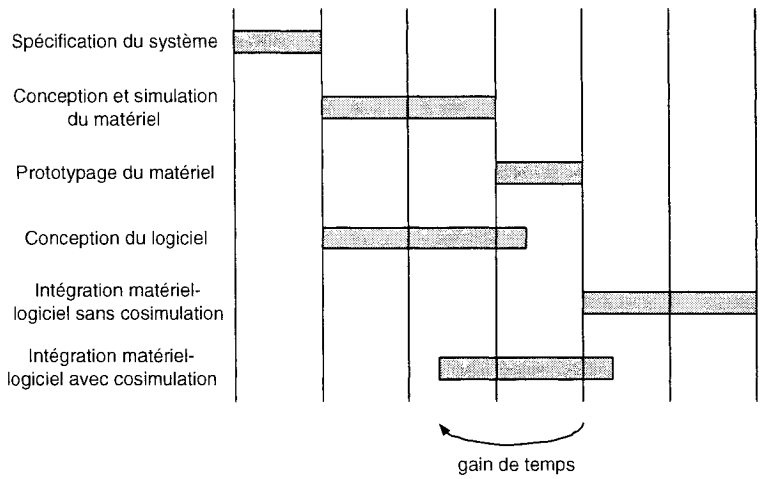

Figure 1.2 Influence de la cosimulation sur le cycle de développement d'un produit

gies classiques), mais il pose aussi des problèmes importants. Au niveau de la spécification et de la modélisation tout d'abord, car il est difficile de décrire le système logiciel/matériel au complet dans un langage unique, indépendamment des futurs choix d'architecture et de partitionnement, et sans influencer ces derniers. [Jerraya et Nicolescu (2003)] souligne l'importance de la spécification/modélisation dans une méthodologie de codesign et affirme qu'il n'existe aujourd'hui aucun langage capable de satisfaire tous les besoins, et en particulier de modéliser correctement à tous les niveaux d'abstraction. Puisqu'il est presque admis qu'aucun langage ne pourra jamais tout faire, l'approche gagnante est donc peut-être de développer un nouveau langage prévu dès le début pour être interconnecté à d'autres langages dans un environnement multilangages [ESys.Net (2005)].

Un autre gros problème du codesign se situe au niveau de la validation des essais de partitionnement, car on ne dispose pas toujours de bonnes métriques de performance. Il est donc important de développer des moyens de quantifier la performance tout au long du processus de codesign, en utilisant par exemple du profilage [Dubois *et al.* (2005)].

Enfin, le codesign pose des problèmes liés à l'exploration architecturale et à la re-

cherche d'un choix de partitionnement optimal[1]. En plus du temps consommé en cycles d'exploration architecturale avant d'arriver à la solution optimale, puisque la nature (logicielle ou matérielle) des blocs fonctionnels du système est potentiellement modifiée à chaque nouvel essai, cela peut nécessiter beaucoup de travail additionnel (double codage de chaque bloc fonctionnel). Divers travaux se penchent sur ces problèmes et tentent de leur apporter des solutions. En particulier, [SPACE (2005)] propose une plateforme de codesign en SystemC [SystemC (2005)] qui, d'une part, facilite l'exploration architecturale en permettant le passage des modules en logiciel ou en matériel à partir d'un codage unique et, d'autre part, offre une approche par raffinement progressif avec un raffinement automatique des communications.

Ces travaux visent à faciliter ou accélérer la conception des SoC. Pour la plupart des méthodologies de conception, une partie des étapes est indépendante du type de puce et de la technologie de fabrication visés. En revanche, les dernières étapes en dépendent systématiquement. Nous allons maintenant regarder les différents types de puces que les fonderies sont aujourd'hui capables de produire.

1.3 Les types de puces

Aujourd'hui, il y a trois grandes classes de puces destinées à la production des SoC : les circuits intégrés spécifiques à une application (en anglais ASIC pour Application-Specific Integrated Circuit), les ASIC structurés et les puces à architecture programmable. Aucune d'entre elles n'est absolument meilleure que les autres, et le choix de l'une sur l'autre doit se faire en fonction du projet entrepris.

[1]Ou tout au moins relativement optimal, puisque pour garantir un choix optimal, il faudrait tester toutes les architectures possibles, et comme le nombre d'architectures augmente exponentiellement avec le nombre de blocs fonctionnels, en pratique, on ne testera jamais tous les cas.

Il faut alors considérer plusieurs critères :

- performance : par exemple la latence maximum ou bien la fréquence d'horloge ou le débit minimum qu'il faut achever ;
- consommation : la puissance consommée ;
- temps d'accès au marché vs. temps de conception et production : le temps dont on dispose pour mener le project à terme par rapport au temps lié d'une part à la conception de la puce et d'autre part à sa fabrication ;
- coût unitaire : le coût de chaque puce ;
- quantité : le nombre de puces ;
- coûts d'ingénierie non récurrents (en anglais NRE costs pour Non Recurring Engineering costs) : somme d'argent chargée par le fondeur pour chaque nouveau design de puce, notamment pour les coûts des masques.

Chaque type de puce offre un certain compromis entre ces différents critères. Nous allons maintenant étudier chaque catégorie de puces un peu plus en détails.

1.3.1 ASIC

De nos jours, les fondeurs proposent des ASIC qui, dépendamment de la technologie, possédent typiquement une ou deux couches de poly (de l'anglais polycrystaline silicon) et de quatre à dix couches métalliques (aluminium ou, de plus en plus, cuivre). Même si on se concentre dans le présent mémoire sur l'électronique numérique, on notera qu'il est possible de fabriquer des circuits ASIC mixtes combinant des blocs analogiques et des blocs numériques.

1.3.1.1 ASIC sur mesure

Ces puces (en anglais full custom ASIC) sont la version la plus personnalisée des ASIC. Chaque élément est dessiné sur mesure et toutes les couches sont gravées sur mesure.

Avantages : performance, consommation, prix unitaire (surface de silicium utile réduite).
Inconvénients : coûts non récurrents très élevés, temps de conception et de fabrication très longs.

Les ASIC sur mesure sont principalement utilisés pour des designs nécessitant le maximum de performance, comme un processeur ou une mémoire. En revanche, lorsqu'il s'agit de SoC, on n'emploiera jamais d'ASIC totalement sur mesure mais, à la limite, un ASIC partiellement sur mesure et partiellement précaractérisé.

1.3.1.2 ASIC précaractérisés

Pour ces puces (en anglais standard cell ASIC ou cell-based ASIC), les circuits doivent cette fois être conçus à partir de cellules élémentaires fournies par le fondeur (sous forme d'une bibliothèque) mais, là encore, toutes les couches sont gravées en fonction du circuit désiré.

Avantages et inconvénients : ce sont les mêmes que pour les ASIC sur mesure, à part qu'ils sont moins prononcés (dans un sens comme dans l'autre).

Les ASIC précaractérisés seront typiquement le choix de prédilection lorsqu'on veut produire un SoC en très grande quantité et qu'on dispose de temps (grand temps d'accès au marché).

Nous omettons ici volontairement les ASIC prédiffusés (en anglais gate array ASIC) car ils perdent énormément en popularité et tendent à être remplacés par les ASIC structurés.

1.3.2 ASIC structurés

Le but des ASIC structurés (en anglais structured ASIC ou platform ASIC) est de proposer une puce ayant des performances de l'ordre de celles des ASIC précaractérisés, tout en réduisant substantiellement le temps de conception. Pour cela, les fondeurs préfabriquent des puces contenant plusieurs tranches de silicium dites plateformes. Ces tranches plateformes intègrent divers modules, afin de pouvoir adresser une grande variété d'applications. Puis, à travers la personnalisation de juste quelques couches métalliques, il est possible de configurer et d'interconnecter les tranches afin d'obtenir une puce ayant les fonctionnalités voulues [Lipman (2004)] [Khalilollahi (2004)]. Afin de permettre aux concepteurs de puces de tirer profit des ASIC structurés, les fondeurs fournissent en fait non seulement les puces, mais aussi une méthodologie et un ensemble d'outils associés.

Par rapport aux ASIC précaractérisés, les ASIC structurés ont comme avantage de réduire grandement à la fois le temps et les coûts liés à la conception et à la fabrication de la puce. En revanche, ils présentent une bien moins bonne densité d'intégration, et ont donc un prix unitaire plus élevé. Alors que performance et densité d'intégration atteindraient un très honnête 80% de ces mêmes caractéristiques pour des ASIC précaractérisés, les coûts seraient quant à eux diminués de 80% [Zakharia (2003)].

Les ASIC structurés seront donc typiquement choisis lorsqu'on veut produire des SoC en quantité moyenne ou quand on veut produire de grosses quantités mais qu'on n'a pas le temps de développer un ASIC précaractérisé.

1.3.3 Puces à architecture programmable

Ces circuits, également appelés circuits à réseaux logiques programmables, sont des circuits dont l'architecture logique interne peut être programmée, voire reprogrammée, après leur fabrication chez le fondeur. Ils permettent de réaliser n'importe quelle fonction logique en reliant, par des connexions programmables, des blocs programmables eux aussi contenant des fonctions logiques élémentaires.

On trouve aujourd'hui deux principales familles de circuits logiques programmables : les CPLD (Complex Programmable Logic Device) et les FPGA (Field Programmable Gate Array). Indépendamment de leur appartenance à l'une de ces deux familles, les puces à architecture programmable présentent les mêmes avantages et inconvénients.

Avantages : coûts de conception et surtout[1] de fabrication peu élevés ; temps d'accès au marché court.
Inconvénients : prix unitaire élevé dû à une faible densité d'intégration ; performance très inférieure à celle des ASIC ; consommation plus importante.

Une caractérisque intéressante des puces à architecture programmable est la capacité de reprogrammation. En effet, dépendamment de la technologie employée, ces puces pourront être programmées une seule fois ou bien plusieurs fois.

1.3.3.1 CPLD

Les CPLD combinent plusieurs matrices logiques programmables ou PLA (de l'anglais Programmable Logic Array) en périphérie de la puce avec une matrice d'in-

[1]Surtout de fabrication, car il n'y pas de coûts d'ingéniérie non récurrents, la spécificité des puces n'étant pas ici dans ce qui est gravé dans silicium, mais dans la façon dont les éléments programmables vont être utilisés dans chaque puce.

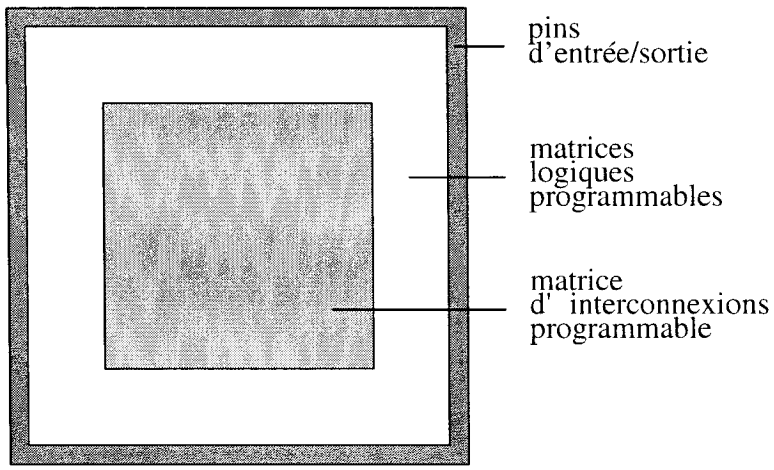

Figure 1.3 Structure d'un CPLD

terconnexions programmable au centre.

Chaque PLA est une matrice logique à réseaux ET et OU programmables, capable de réaliser n'importe quelle fonction logique sous forme de somme de produits. Les blocs peuvent ensuite être reliés entre eux à travers la matrice d'interconnexions. Grâce à cette organisation, les délais de propagation entre les blocs à travers la matrice d'interconnexions sont fixes, et connus avant les étapes de placement et routage. En conséquence, ces phases du cycle de conception sont faciles et donc peu gourmandes en temps.

La grande majorité des CPLD sont reprogrammables.

1.3.3.2 FPGA

Plutôt que de définir des zones bien séparées de logique programmable et d'autres d'interconnexions programmables, on trouve dans un FPGA de nombreux blocs logiques programmables simples éparpillés dans un tissu d'interconnexions programmable.

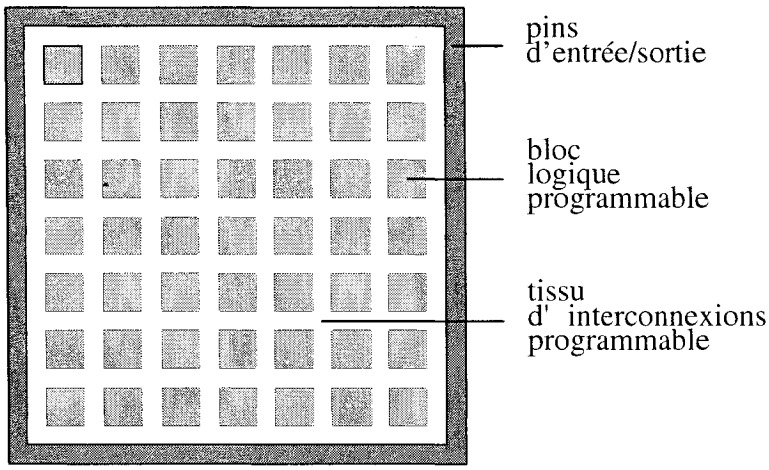

Figure 1.4 Structure d'un FPGA

Avec ce type d'organisation, on gagne beaucoup en densité d'intégration sur les CPLD et on peut tirer meilleur profit des ressources disponibles. En revanche, les phases de placement et routage sont gourmandes en temps, surtout si on souhaite exploiter les possibilités de la puce au maximum, et les délais dus aux interconnexions ne peuvent être connus qu'après ces étapes.

Du point de vue de la reprogrammabilité, on trouve des deux cas dans les FPGA. D'un côté, par exemple, les FPGA utilisant la technologie anti-fusible au niveau des interconnexions ne sont programmables qu'une seule fois, mais c'est cette technologie qui offre la meilleure densité d'intégration et les meilleures performances. D'un autre côté, les FPGA utilisant la technologie SRAM pour les interconnexions sont reprogrammables. Cette technologie facilite en outre l'intégration de mémoire RAM dans le circuit, i.e. directement sur la puce [Rabasté (2002)] [FPGA-FAQ (2005)].

Typiquement, les FPGA seront utilisés pour les projets ayant d'importantes contraintes de temps (temps d'accès au marché petit) et/ou nécessitant la production d'un petit nombre de puces.

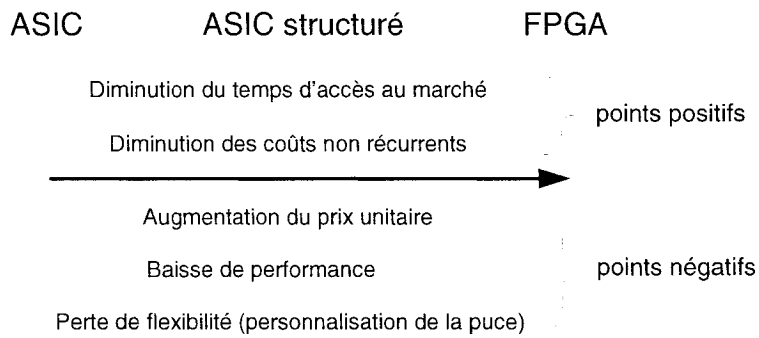

Figure 1.5 Récapitulatif des caractéristiques des types de puces

1.3.4 Récapitulatif

Pour conclure sur les différents types de puces, la figure 1.5 présente un récapitulatif de leurs avantages et inconvénients. Plus exactement, elle montre les tendances observées lorsqu'on va des ASIC vers les FPGA en passant par les ASIC structurés. Si on réfléchit dans le sens opposé, migrer d'un FPGA vers un ASIC permet un gain de performance et une diminution du coût unitaire des puces au prix d'importants temps et coûts de conception et d'ingéniérie non récurrents.

Il est difficile de comparer numériquement les ASIC et les FPGA en termes de performances, notamment parce que les métriques utilisées diffèrent d'une catégorie à l'autre et d'un vendeur à l'autre. Si l'on souhaite malgré tout se risquer à donner des chiffres : fin 2005, pour une technologie 90 nm, la fréquence de l'horloge dans un design FPGA atteint 500 MHz alors qu'elle peut aisément dépasser 1 Gz dans un design ASIC ; le nombre de portes pour un ASIC peut atteindre 100 millions alors que les plus gros FPGA proposent l'équivalent d'un peu plus de 2 millions de portes ASIC ; enfin, pour effectuer la même fonction logique, la puissance consommée est 5 à 10 fois supérieure pour un FPGA que pour un ASIC.

On notera que sur le marché hyperdynamique d'aujourd'hui, celui de l'électronique grand public, les critères de choix les plus importants sont un temps d'accès au

marché petit et un prix unitaire faible. Or ces deux données suivent des évolutions opposées. Il n'est donc pas rare pour les concepteurs de réaliser leur circuit d'abord sur des FPGA ou des ASIC structurés avant de migrer leur design vers des ASIC. Ceci permet en effet de pénétrer un nouveau marché plus rapidement et, si le produit marche, de baisser les coûts de production pour les générations suivantes.

1.4 La montée en puissances des FPGA

Si les progrès de la microélectronique ont pour effet d'améliorer les performances de tous les types de puces, c'est dans le domaine des FPGA que l'on voit actellement le plus de progrès. En conséquence, le niveau de performance des FPGA se rapproche de plus en plus de celui des ASIC. L'écart au niveau de la densité d'intégration se réduit lui aussi de plus en plus. Ceci a pour effet de réduire les coûts unitaires qui, comme nous l'avons vu précédemment, représentent le principal désavantage des FPGA. Les constructeurs de solutions reprogrammables annoncent même depuis quelque temps des solutions FPGA moins chères que les ASIC structurés [Krishnan et Thirumalai (2004)].

Par ailleurs, bien qu'il existe des différences entre les ASIC et les FPGA, ces derniers bénéficient de plus en plus des outils et méthodologies initialement développés et utilisés pour les ASIC, comme la synthèse physique [Southard (2004)]. Déjà en 2003, 53% des concepteurs de SoC n'utilisaient que des FPGA, alors que seulement 13% n'en utilisaient pas du tout [Celoxica (2003)]. Aujourd'hui, tout le monde semble être d'accord pour dire que l'utilisation des FPGA continue d'augmenter.

Avec d'un côté une forte amélioration de leurs points faibles (i.e. des performances et des coûts unitaires se rapprochant de ceux des ASIC) et un nombre croissant d'outils et méthodologies utilisables, et avec d'un autre côté une consolidation de leurs avantages évoqués plus tôt, les FPGA apparaissent comme la solution

gagnante du moment pour la conception de systèmes sur puce. Au cours des dix dernières années, les progrès qu'ils ont connus ont dépassé les prévisions de la loi de Moore, et la compétition est très féroce [Morris (2005)].

Une autre raison du succès croissant des FPGA est l'apparition récente d'un nouveau produit : le FPGA plateforme.

1.5 FPGA plateformes

Les FPGA plateformes sont des FPGA sur lesquels, en plus des blocs logiques programmables et du réseau d'interconnexions programmable, ont été ajoutées en dur (de l'anglais hard core) des propriétés intellectuelles (en anglais IP pour Intellectual Property), comme par exemple un microprocesseur. Ces modules existant en dur ne font pas partie de l'architecture programmable, et il n'est donc pas possible de les remplacer par autre chose. En revanche, ils ne consomment aucun bloc logique, et ils sont plus petits (en termes de surface occupée) et plus performants que le seraient leurs équivalents en logique programmable.

Afin que les concepteurs puissent tirer profit au maximum de ces puces, leurs fabricants offrent des outils et méthodologies associés, incluant une bibliothèque d'IP. Ces outils permettent d'adopter une approche plateforme, et les concepteurs peuvent ainsi, à partir d'un système standard réalisé à partir des IP fournies, venir greffer leurs modules personnalisés.

Comme il s'agit de produits encore récents, il n'y a pas encore de véritable consensus au niveau de la terminologie les concernant. Dans ce mémoire, les puces seront toujours appelées FPGA plateformes (platform FPGA en anglais). Pour désigner les systèmes qu'elles contiendront, nous préférerons en revanche parler de systèmes sur puce programmable (de l'anglais SOPC pour System On Programmable Chip).

Nous allons maintenant passer en revue les fabricants de FPGA plateformes, et présenter leurs offres (puces et suite d'outils) respectives.

1.5.1 Triscend

Triscend, Corp. fut la première compagnie, en août 2000, à lancer un FPGA plateforme, le A7, qui incorpore le très populaire microprocesseur ARM7TDMI de la compagnie ARM Ltd [ARM (2005)]. Depuis, Triscend a été rachetée (en mars 2004) par Xilinx, Inc. [Xilinx (2005)] et a cessé de produire ses puces et de fournir son outil, FastChip. Nous n'en parlerons donc pas davantage.

1.5.2 Altera

La compagnie Altera [Altera (2005)] a lancé son FPGA plateforme, nommé Excalibur, en octobre 2001. Du côté logique programmable, l'Excalibur repose sur les FPGA Altera APEX 20KE. Du côté IP en dur, Altera a choisi d'intégrer un système complet fonctionnel sur la puce. Les figure 1.6 montrent l'architecture des puces Excalibur, où l'on voit clairement apparaître les deux zones de la puce : la zone processeur embarqué (embedded processor stripe dans la terminologie Altera) et la zone logique programmable.

On trouve donc à l'intérieur de la puce, en dur : un bloc processeur ARM922T, de la mémoire SRAM, des minuteries, un contrôleur d'interruptions, un émetteur-récepteur asynchrone universel (en anglais UART pour Universal Asynchronous Receiver-Transmitter) et une boucle à vérouillage de phase. Ces éléments, qui forment le cœur de la zone processeur embarqué, sont interconnectés grâce à deux instances du bus sur puce ARM AHB. La puce intègre également un contrôleur de mémoire SDRAM externe et un bus EIB (Extension Bus Interface) permettant de

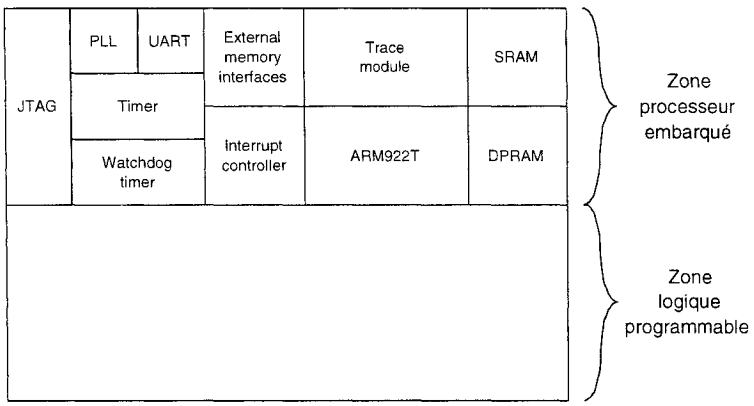

Figure 1.6 Architecture de l'Excalibur

connecter au système des éléments externes tels qu'une mémoire flash.

Il est intéressant de noter que le système contenu dans la zone processeur embarqué est capable de fonctionner indépendamment de ce qu'on décide de mettre en logique programmable, ce qui présente évidemment des avantages et des inconvénients, sur lesquels nous reviendront plus tard.

Une autre particularité de la puce Altera vient de sa partie logique programmable, qui incorpore, à l'instar des APEX 20KE, des ESB (Embedded System Block) permettant d'implémenter différents types de blocs mémoires, comme par exemple des files (FIFO).

1.5.3 Xilinx

La compagnie Xilinx est arrivée la dernière sur le marché des FPGA plateformes quand elle a sorti le Virtex-II Pro, en mars 2002. Ces puces reposent sur le Virtex-II pour ce qui est de la logique programmable et, du côté des IP en dur, elles intègrent un ou deux microprocesseurs PowerPC 405 de IBM Corp. [IBM (2005)] et un nombre variable d'émetteurs-récepteurs multigigabit (en anglais MGT pour Multi-Gigabit Transceiver). La figure 1.7 illustre l'architecture du Xilinx Virtex-II

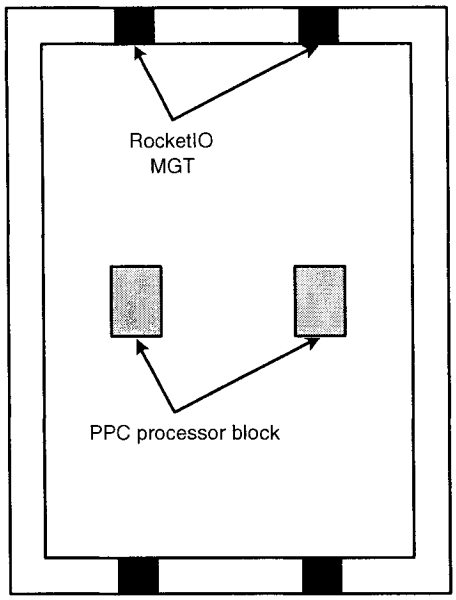

Figure 1.7 Architecture du Virtex-II Pro

Pro.

L'approche de Xilinx est radicalement différente de celle d'Altera. En effet, si le concepteur ne crée pas de système, le Virtex-II Pro n'est pas capable de faire quoi que ce soit. Autrement dit, le concepteur a la liberté d'utiliser à sa guise ce que la puce met à sa disposition.

Tout comme le Virtex-II, le Virtex-II Pro intègre également des multiplicateurs 18 bits par 18 bits, qui sont très utiles pour des applications telles que le traitement de signal.

1.5.4 Comparaison

Avant de tenter une comparaison entre les produits d'Altera et de Xilinx, il est important de préciser que la guerre que se mènent les deux compagnies est très féroce, et qu'à travers la quantité d'informations plus ou moins pertinentes et

à cause de la façon dont les constructeurs les manipulent, il est difficile de s'y retrouver. En particulier, comme le mentionne [Morris (2005)], la bataille sur le nombre de portes est une bataille perdue d'avance...pour le client potentiel en quête de vérité. Nous nous contenterons donc ici de comparer ce qui est réellement comparable.

Premièrement, et de façon évidente, la principale distinction entre l'Excalibur d'Altera et le Virtex-II Pro de Xilinx se fait au niveau des approches diamétralement opposées que les compagnies ont retenues. D'un côté, Altera a décidé de prendre son APEX 20KE et de lui accoler un système complet, pouvant potentiellement fonctionner de manière totalement indépendante du reste de la puce, et non modifiable. D'un autre côté, Xilinx a préféré modifier son Virtex-II pour y éparpiller quelques modules supplémentaires, complètement dépendants de ce qui est programmé sur le reste de la puce.

Avec Xilinx, on a donc une plateforme entièrement personnalisable. Le concepteur pourra par exemple décider de n'utiliser qu'un PowerPC, ou de relier les modules de son système à l'aide d'un réseau sur puce au lieu d'utiliser les IP du bus sur puce IBM CoreConnect fournis avec la suite d'outils. Ceci signifie aussi qu'il faudra passer du temps à créer une plateforme de base avant de pouvoir utiliser la puce. Néanmoins, le Virtex-II Pro est une meilleure solution que l'Excalibur si l'on souhaite par exemple tester différents moyens d'interconnexions.

En revanche, avec Altera, on dispose directement d'une puce contenant un système fonctionnel et utilisable, mais aussi beaucoup moins personnalisable, ce qui en soit présente assez peu d'avantages. Ce qui est beaucoup plus intéressant, c'est que ceci permet de reconfigurer complètement la logique programmable en cours d'utilisation, puisque le système dur ne sera pas affecté du tout par cette reprogrammation. En comparaison, le Virtex-II Pro n'est capable que de reconfiguration

partielle. Pour des travaux comme par exemple le partitionnement logiciel/matériel dynamique [Stitt *et al.* (2003)] nécesitant la reconfiguration de la logique programmable en cours d'exécution, l'Excalibur apparaît donc comme la solution la plus avantageuse.

Deuxièment, il y a une différence importante au niveau des microprocesseurs embarqués. Sur la puce Altera, il s'agit d'un ARM922T, au cœur duquel on trouve un noyau de microprocesseur à usage général ARM9TDMI, à architecture Harvard 32 bits, et possédant un pipeline à 5 étages. Il contient également deux mémoires caches (instructions et données) et une unité de gestion de mémoire (en anglais MMU pour Memory Management Unit), et est cadencé à 200 MHz.

Du côté Xilinx, on trouve un (ou plusieurs) IBM PowerPC 405D5, reprenant les caractéristiques du ARM922T (architecture Harvard 32 bits, pipeline 5 étages, deux caches, MMU), capable d'atteindre une fréquence d'horloge de 400 MHz, et offrant quelques suppléments tels qu'un multiplicateur matériel. Il est d'une manière générale plus performant que son concurrent.

Troisièmement, ces puces diffèrent de part le type et la quantité de mémoire qu'elles intègrent. Dans l'Excalibur, on a de la SRAM réservée à la zone processeur, de la SRAM à double port partagée entre la zone processeur embarqué et la logique programmable, et des blocs de mémoire (les ESB) dans la zone logique programmable. Les ESB peuvent au choix être utilisés comme RAM, DPRAM, ROM ou CAM.

Dans le Virtex-II Pro, on a seulement des blocs de mémoire (Block SelectRAM+) distribués à travers le réseau de logique programmable et pouvant être utilisés comme RAM ou comme DPRAM. Au final, aucune des solutions n'est vraiment meilleure que l'autre, l'absence de flexibilité au niveau des SRAM que contient l'Excalibur étant contrebalancée par la grande flexibilité de ses ESB.

Tableau 1.1 Comparaison des puces Altera et Xilinx

Altera Excalibur EPXA10 (APEX 20KE)	Xilinx Virtex-II Pro XC2VP100 (Virtex-II)
Zone processeur embarqué contenant un sous-système complet fonctionnel	
1 ARM922T ARM AMBA	2 IBM PowerPC 405D5 – multiplicateur matériel IBM CoreConnect
1 bloc 256 ko SRAM 2 blocs 64 ko DPSRAM 160 blocs ESB, 2 kb par bloc Total : environ 700 ko de mémoire	444 blocs Block SelectRAM+, 18 kb par bloc Total : environ 1 Mo de mémoire
	444 multiplicateurs 18 bits par 18 bits

Quatrièmement, le Virtex-II Pro propose quelques blocs fonctionnels supplémentaires. Cette puce sera donc avantageuse pour certaines classes d'applications telles que le traitement de signal (multiplicateurs 18 bits par 18 bits) ou les réseaux (MGT).

Le tableau 1.1 résume les différences existant entre les puces Altera et Xilinx. Les caractéristiques quantitative citées ici sont celles du modèle le plus performant de chaque constructeur [Altera (2005)] [Xilinx (2005)].

On notera qu'on a pour l'instant comparé les deux solutions uniquement sur les différences matérielles au niveau des puces. D'autres éléments font pourtant partie intégrante de la solution globale, comme c'est notamment le cas pour la bibliothèque d'IP fournie. Les bibliothèques d'IP sont à peu près équivalentes pour les deux constructeurs, mais une différence émerge pourtant, au niveau des processeurs logiciels (en anglais soft processor) développés par chacune des compagnies : le NIOS-II d'Altera qui se connecte sur un bus Altera Avalon, et le MicroBlaze de Xilinx qui se connecte sur un bus IBM CoreConnect OPB.

Les outils de conception sont également importants. Ils font l'objet de la partie

suivante.

1.6 Outils

1.6.1 Xilinx

En plus de son environnement classique de développement sur FPGA ISE (Integrated Software Environment), Xilinx propose EDK (Embedded Development Kit), une suite d'outils de développement plus adaptée aux FPGA plateformes. EDK propose principalement une grande bibliothèque d'IP et XPS (Xilinx Platform Studio), un outil assez complet permettant de concevoir le système, que ce soit la partie matérielle ou la partie logicielle.

Dans XPS, on construit le système en utilisant le principe du glisser-déposer à partir des blocs IP fournis ou créés par l'utilisateur. On peut alors configurer certaines propriétés des blocs insérés et choisir comment les relier entre eux. Les versions récentes proposent plusieurs assistants, permettant par exemple de créer un système de base, de générer une carte d'adressage mémoire ou un script de compilation, ou encore d'importer des IP créées avec ISE. L'outil permet ensuite de synthétiser le code et de le charger dans un FPGA.

1.6.2 Altera

L'environnement de développement Altera, Quartus-II, permet également de concevoir les parties logicielle et matérielle du système embarqué. En outre, l'outil Altera permet, dans une seule et unique vue hiérarchique, et ce sans avoir à passer par un quelconque autre outil, de mélanger des IP complexes provenant des bibliothèques d'IP avec de simples portes logiques, et d'entrer directement des blocs de code

HDL (Hardware Description Language). Cette possibilité de mélanger des blocs à différents niveaux est très pratique et impossible avec les outils Xilinx.

De façon à faciliter et accélérer la création de systèmes, Quartus-II intègre SOPC Builder, un outil permettant de générer automatiquement et facilement des systèmes complets, simples ou complexes, et de configurer les blocs IP du système. SOPC Builder permet également de créer des versions personnalisées du processeur logiciel NIOS-II d'Altera.

1.6.3 Récapitulatif

Si du côté de la puce, l'offre de Xilinx semble meilleure, du côté de la suite de logiciels de développement, la première marche du podium revient à Altera : tout d'abord, la vue hiérarchique dans Quartus-II est extrèmement pratique pour concevoir les systèmes ; ensuite, SOPC Builder est bien plus avancé que l'assistant proposé par XPS ; et enfin, avec Quartus-II, on fait tout à partir du même outil, alors que chez Xilinx, on doit manipuler à la fois ISE et XPS.

Maintenant que nous avons vu ce que proposent les compagnies fabriquant des FPGA plateformes, nous allons nous intéresser à l'utilisation qu'on peut faire de ces produits.

1.7 Applications des FPGA plateformes

Grâce à leur grande versatilité et à leur relative facilité de conception rendue possible par les outils qui les accompagnent, les FPGA plateformes constituent un excellent outil éducationnel. Une fois passé l'investissement - à tarif souvent réduit pour le milieu universitaire, en cartes de développement et outils de conception, il

est possible de réaliser, sans coût additionnel, un grand nombre de projets aussi divers que des applications Internet ou des robots [Hall et Hamblen (2004)].

La puissance de ces puces récentes, associée à la disponibilité d'IP de communications à haute vitesse, permet également de réaliser en temps réel des traitements très complexes. Les FPGA plateformes trouvent donc des applications dans des domaines tels que la réseautique haute vitesse [Brebner (2002)] et le traitement de flux video [Ghozzi *et al.* (2003)] [Kordasiewicz et Shirani (2004)], ou bien encore le contrôle temps réel [Sancho-Pradel *et al.* (2002)] et l'imagerie médicale [Li *et al.* (2004)].

Les FPGA, et à fortiori les FPGA plateformes, sont également extrèmement utilisés pour faire du prototypage : le prototypage sur FPGA fait en effet partie intégrante du flot de conception ASIC pour près de 70% des concepteurs de ASIC [Celoxica (2003)]. Néanmoins, plus le projet compte de portes, plus son prototypage pose de problèmes [Mahmud (2004)]. Mais avec les récents développements dans le domaine des FPGA et l'apparition de nouveaux outils [Blyler (2004)], on peut désormais prototyper des designs de plus en plus gros avec un nombre de FPGA de plus en plus réduit, et de nouvelles cartes de prototypages très performantes font leur apparition sur le marché [Rizzatti (2004)].

Les FPGA sont aussi devenus, avec l'augmentation du nombre de portes logiques qu'ils contiennent, des plateformes très pratiques et très bon marché pour des applications multiprocesseurs. Mais comme un FPGA peut contenir une grande variété d'architectures matérielles, il peut devenir très délicat d'adapter le système d'exploitation temps réel (en anglais RTOS pour Real Time Operating System), surtout dans le cas où le design comprend plusieurs microprocesseurs. Pour pallier ce problème, on peut développer un RTOS configurable et une technique de coconfiguration [Takada *et al.* (2003)] automatique du RTOS en fonction de l'architecture

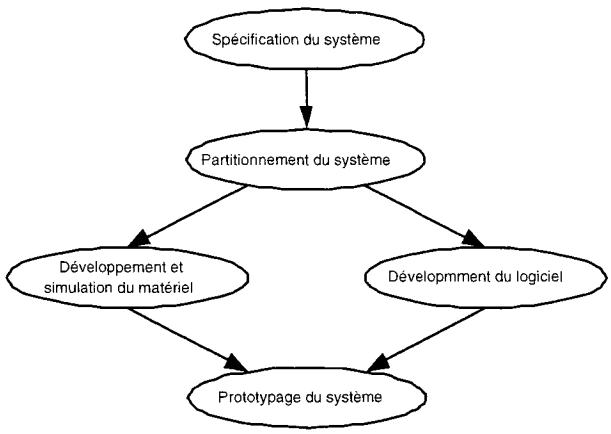

Figure 1.8 Flot de conception classique pour FPGA plateforme

matérielle.

De par leur nature mixte logicielle et matérielle, les FPGA plateformes sont d'excellents supports de codesign. Aujourd'hui, les outils des deux constructeurs permettent de tirer profit de méthodologies ou de techniques de conception telles que la réutilisation de la propriété intellectuelle ou l'approche plateforme, mais il reste encore des améliorations à faire pour accélérer le cycle de conception, ce qui est principalement lié à deux limitations importantes des outils actuels : premièrement, les temps de synthèse et de placement/routage augmentent très rapidement (de manière non linéaire) avec la complexité et la taille du design ; et deuxièmement, ces outils effectuent la synthèse à partir de modèles HDL et ne permettent pas de profiter des avantages de la conception à haut niveau d'abstraction.

La figure 1.8 illustre un flot de conception classique sur FPGA plateforme avec les outils actuels. Ces limitations peuvent être contournées en utilisant un principe déjà utilisé dans le monde des ASIC : la cosimulation.

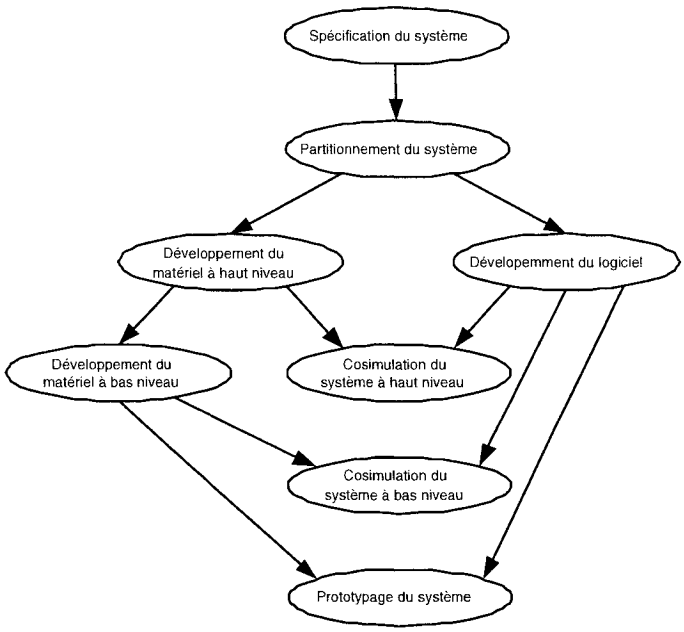

Figure 1.9 Flot de conception pour FPGA plateforme avec cosimulation

1.8 La covérification avec Seamless

Seamless CVE (Co-Verification Environment) est un outil de cosimulation/covérification développé par Mentor Graphics Corp. [Mentor (2005)a] permettant de synchroniser un simulateur logiciel avec un simulateur matériel. Ici il s'agira de XRAY, un simulateur de jeu d'instructions (en anglais ISS pour Instruction Set Simulator) et débogueur logiciel, et de Modelsim, un simulateur matériel. De façon à cosimuler le FPGA plateforme, on a juste besoin des modèles HDL et on évite de passer à travers les étapes, gourmandes en temps, de synthèse et placement/routage.

1.8.1 Épargner le temps gaspillé en synthèses et placements/routages successifs

La figure 1.9 montre à quel moment la cosimulation intervient dans le processus de conception.

Pendant la cosimulation, on bénéficie de tous les avantages des deux débogueurs, et on peut à n'importe quel moment regarder ce qui se passe et en matériel et en logiciel (avec la possibilité d'avancer instruction par instruction, de changer la valeur d'un registre, etc.) et ainsi vérifier le bon fonctionnement du système.

On peut donc faire plusieurs boucles de cosimulation et détecter des erreurs ou modifier le design avant même de le synthétiser pour la première fois. Afin que ceci soit rentable, i.e. que ça apporte un gain de temps, il faut néanmoins s'assurer que les durées de cosimulation restent aussi courtes que possible.

1.8.2 Accélérer la cosimulation

Comme l'outil de cosimulation synchronise les simulateurs matériel et logiciel, la vitesse de la cosimulation est dictée par la simulation la plus lente (matérielle). Seamless propose des modes d'optimisation accélérant la simulation de manière significative. Ces optimisations sont en partie rendues possibles grâce à l'existence du serveur de mémoire cohérente : lorsqu'une mémoire est déclarée comme optimisable, son contenu est placé dans un tableau en mémoire géré par le serveur de mémoire et auquel matériel et logiciel peuvent tous deux accéder de manière complètement indépendante mais cohérente [Mentor (2005)b]. La figure 1.10 illustre le fonctionnement du serveur de mémoire cohérente.

Grâce à ce mécanisme, Seamless peut alors supprimer l'exécution de certains cyles matériels : par exemple, si le logiciel veut accéder à une variable, au lieu d'aller la chercher dans la mémoire simulée en matériel, Seamless peut tout simplement prendre la copie que le serveur de mémoire cohérente gère.

Seamless propose trois types d'optimisations, dont deux sont la conséquence directe de l'utilisation du serveur de mémoire cohérente :

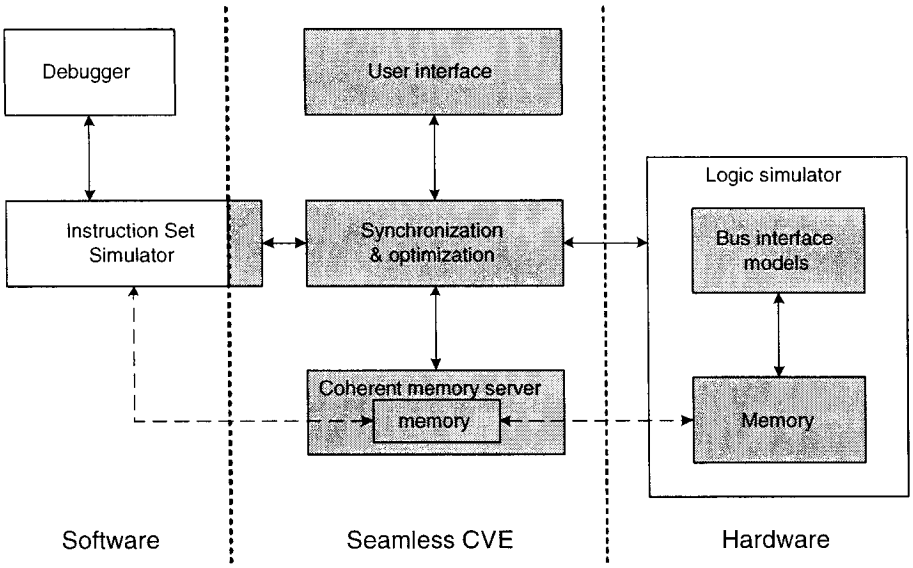

Figure 1.10 Serveur mémoire Seamless

- optimisation des lectures d'instructions : suppression des cycles matériels pour toutes les lectures d'instructions ;
- optimisation des accès aux données : suppression des cycles matériels pour les accès aux données dans certaines zones définies de l'espace mémoire ;
- optimisation du temps : désynchronisation des simulations matérielle et logicielle et suppression de tous les cycles matériels sauf occasionnellement.

Afin de permettre les optimisations au niveau des accès mémoire, il faut que les mémoires du design soient compatibles avec le serveur de mémoire de Seamless. L'outil PureView de Denali Software, Inc. [Denali (2005)] fourni avec Seamless permet de générer de tels modèles, pouvant être pleinement utilisés par Seamless, pour la plupart des mémoires répandues sur le marché.

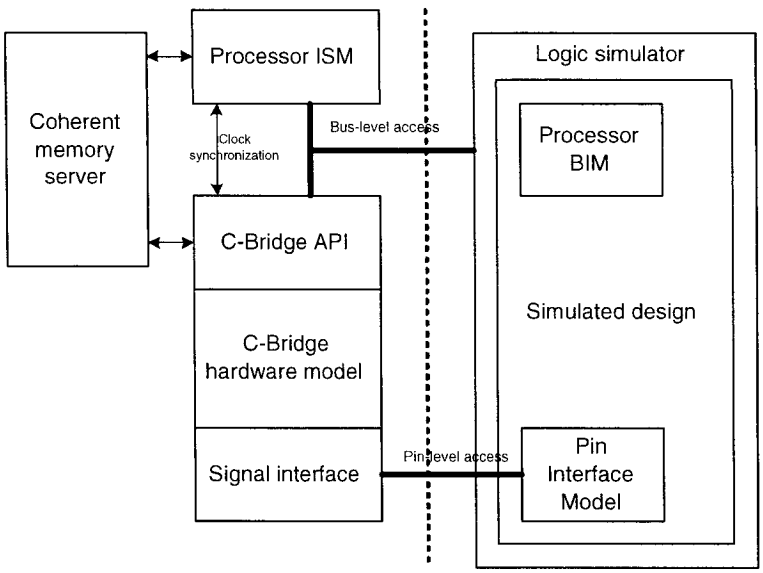

Figure 1.11 C-Bridge dans Seamless

1.8.3 Gérer les modèles matériels décrits dans des langages à haut niveau d'abstraction

Seamless inclut une fonctionnalité appelée C-Bridge permettant de simuler et d'interfacer avec le reste de la cosimulation des modèles matériels écrits en C ou en C++ (y-compris SystemC).

Comme il apparaît sur la figure 1.11, les modules C-Bridge peuvent communiquer à deux niveaux :

- au niveau bus, auquel cas le module est juste connecté au processeur à travers un bus transactionnel ;
- au niveau pins, auquel cas le module C-Bridge possède un adapteur HDL qui doit être instancié dans la partie matérielle de la plateforme, et qui connecte les signaux d'entrée/sortie du module à n'importe quels autres signaux du design.

L'utilisation de Seamless dans le flot de conception d'un FPGA plateforme permettrait donc, grâce aux deux niveaux de communications C-Bridge et au support non

seulement du C/C++ mais aussi des langages HDL (VHDL, Verilog), de concevoir et intégrer de nouveaux modules dans le système par raffinement progressif.

D'autres solutions peuvent être utilisées en dehors de Seamless pour accélérer le cycle de développement sur FPGA.

1.8.4 Autres solutions

Tout d'abord, d'autres outils de cosimulation visant le Virtex-II Pro existent - ou pour certains ont existé. La compagnie Innoveda, Inc. avait annoncé le support du Virtex-II Pro dans V-CPU, mais c'est désormais la compagnie Summit Design, Inc. [Summit (2005)] qui offre cet outil. La compagnie Endeavor Intertech Corp. [Endeavor (2005)] quant à elle a sorti CoSimple peu après l'arrivée de la puce, mais il semblerait aujourd'hui que cette compagnie ait cessé toute activité.

Une approche différente a été adoptée par la société Emulation and Verification Engineering (EVE), SARL [EVE (2005)] avec leur solution de vérification assistée par matériel ZeBu (Zero Bugs). Leur produit permet non seulement de tirer à la fois profit des avantages de l'émulation et du prototypage sur FPGA, mais aussi en quelque sorte de cosimuler un système à partir de la carte de développement, ce qui permet une accélération du côté matériel sans pour autant compromettre l'observabilité et la contrôlabilité.

Il est possible d'accélérer le cycle de développement sur FPGA en permettant l'exécution de modèles matériels décrits à haut niveau d'abstraction et donc non synthétisables. En utilisant les PowerPC inutilisés du Virtex-II Pro pour exécuter ces modèles haut niveau, et en les reliant au reste du système à travers un adapteur matériel, on peut intégrer au système des modèles matériels n'ayant pas encore été développés au niveau d'abstraction traditionnellement nécessaire [Ohba et Takano

(2004)]. Même si le nombre de PowerPC présents sur une puce est très limité (deux maximum par puce au lieu de quatre maximum par puce comme initialement prévu par Xilinx), l'idée est très intéressante car on pourrait très bien faire la même chose avec des microprocesseurs logiciels comme MicroBlaze.

Nous avons dans ce chapitre rappelé les concepts généraux liés aux systèmes sur puce et au codesign, et en particulier à l'étape de cosimulation/covérification. Nous avons également défini les FPGA plateformes, précisé leur place par rapport à autres puces, donné certaines de leurs applications typiques et présenté brièvement les outils qui les accompagnent.

Nous allons maintenant présenter la carte de développement dont nous avons voulu créer un modèle de simulation, ainsi que le design dont nous sommes partis afin d'obtenir notre modèle.

CHAPITRE 2

ARCHITECTURE DE RÉFÉRENCE CIBLE

Un des buts premiers de ce travail étant de pouvoir recréer dans un environnement de simulation les possibilités offertes par l'utilisation de la carte AP100 disponible dans nos laboratoires, nous avons suivi les spécifications de cette dernière lors du développement de notre modèle. Nous allons donc présenter cette carte et voir ce qu'il est possible de faire avec.

2.1 Carte de développement

La carte AP100 est une carte au format PCI prévue pour le prototypage et le développement de systèmes sur puce programmable et développée par la compagnie Amirix [Amirix (2005)]. Elle est équipée d'un FPGA plateforme Xilinx Virtex-II Pro et de divers mécanismes permettant de faciliter le développement de tels systèmes.

2.1.1 Principaux composants

La figure 2.1 présente un schéma blocs de la carte AP100. Il y apparaît très clairement que le Virtex-II Pro tient la place centrale dans le design de la carte, et que divers éléments ont été greffés autour de la puce afin d'augmenter les possibilités offertes par la carte. Nous reviendrons plus tard sur ce qui est dans le FPGA et allons pour l'instant décrire ce qui est autour de la puce.

Directement reliée au Virtex-II Pro, on trouve tout d'abord 64 MB de mémoire vive de type DDR SDRAM. Cette mémoire est accessible au processeur embarqué

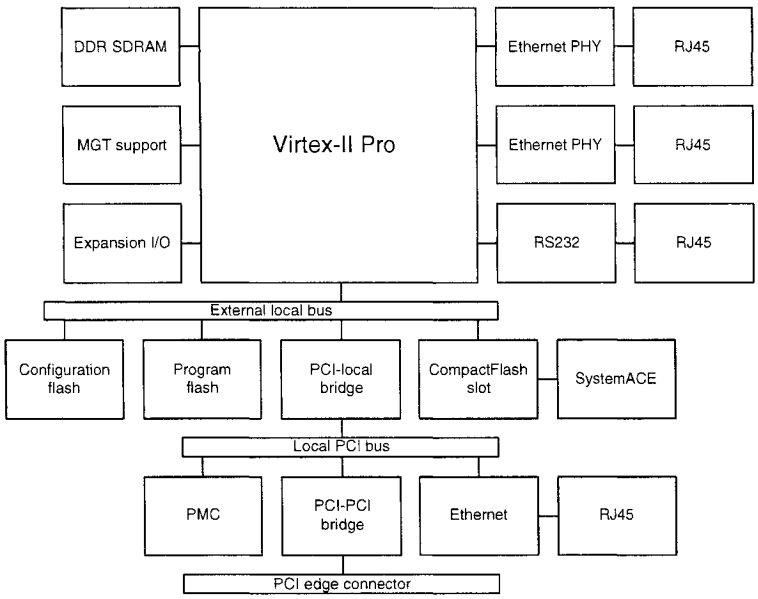

Figure 2.1 Schéma blocs de la carte AP100

dans le FPGA et permet l'exécution de n'importe quel programme de taille raisonnable, ce qui aurait été impossible en utilisant seulement de la BRAM. On trouve également tout le matériel et la connectique nécessaires à l'utilisation éventuelle des émetteurs-récepteurs multigigabit, de deux interfaces Ethernet gigabit et d'un port sériel RS-232. Le choix est laissé au concepteur de système sur puce d'utiliser ou non chacun des éléments cités précédemment, mais il faut ajuster la logique programmable en conséquence. Il est également possible de connecter n'importe quoi directement sur le FPGA en utilisant les entrées/sorties inutilisées.

Au travers du bus local externe, le Virtex-II Pro est connecté à deux puces de mémoire flash (configuration et programme) et à un lecteur de mémoire CompactFlash. La mémoire flash configuration contient les données de configuration du FPGA utilisées par défaut lors de la mise sous tension de la carte. La mémoire flash programme contient quant à elle le code exécutable des outils tels que PPCBoot. On notera que comme elle a une largeur de 16 bits, il est impossible d'exécuter le code directement à partir de cette mémoire, et qu'il faudra auparavant le copier en

mémoire vive (SDRAM). Ces deux mémoires, ainsi que l'éventuelle mémoire CompactFlash insérée dans le lecteur, sont impliquées dans le processus de démarrage de la carte, sur lequel nous revenons dans la prochaine sous-section.

Enfin, au travers du bus local externe et du bus PCI local, le FPGA est connecté à un contrôleur Ethernet 10/100 permettant de relier la carte à un réseau informatique. Il est aussi possible de connecter une carte fille.

2.1.2 Méthodes de démarrage

Par défaut, quand la carte AP100 est livrée, lors de la mise sous tension de la carte, le Virtex-II Pro est programmé d'après les données correspondant au design baseline présentes dans la mémoire flash configuration. Ensuite, la SDRAM est initialisée à partir des données présentes dans la mémoire flash programme, et en particulier l'outil PPCBoot y est chargé. PPCBoot, aujourd'hui continué sous le nom de U-Boot [U-Boot (2005)], est un micrologiciel qui permet notamment de voir ou modifier le contenu des différentes mémoires, charger des images exécutables et lancer des applications sur le PowerPC.

Il est également possible de démarrer la carte de manière différente. Il y a en fait quatre modes de démarrage, et on peut sélectionner le mode à utiliser par l'intermédiaire de deux cavaliers sur la carte. Nous allons les présenter brièvement.

Mode 00. Dans ce mode, le FPGA est programmé à partir d'une des configurations présentes dans la mémoire flash configuration. La flash configuration est assez grande pour contenir de deux à quatre configurations distinctes, et afin de sélectionner laquelle utiliser, il faut auparavant modifier la valeur d'un octet de sélection lui aussi contenu dans la mémoire flash configuration. C'est dans ce premier mode de démarrage que la carte est réglée à la livraison, l'octet de sélection étant

ajusté de manière à utiliser la configuration dite sans échec, i.e. la configuration correspondant au design baseline.

Mode 01. Dans ce mode, le FPGA est programmé à partir de la configuration sans échec contenue dans la mémoire, et ce quelle que soit la valeur de l'octet de sélection. Il s'agit donc d'un mode sans échec, qui assure de pouvoir revenir à une configuration fonctionnelle même si l'octet de sélection a été modifié mais n'a pas pu être remis à sa valeur par défaut.

Mode 10. Dans ce mode, le FPGA n'est pas programmé à partir de la mémoire flash configuration, mais à partir des données présentes sur la mémoire Compact-Flash insérée dans le lecteur, par l'intermédiaire de SystemACE CF, une solution de configuration de FPGA disponible chez Xilinx. Grâce à ce mode, on peut très facilement charger n'importe quelle configuration dans le Virtex-II Pro sans modifier quoi que ce soit (reprogrammer la mémoire flash configuration) sur la carte et sans utiliser d'outil et de machine hôte.

Mode 11. Dans ce dernier mode, le FPGA est configuré par JTAG directement à partir de la machine hôte. Il faut pour cela obligatoirement être équipé d'un câble spécial, mais il est ensuite extrêmement simple et pratique de reprogrammer le Virtex-II Pro avec n'importe quelle configuration.

2.1.3 Mécanismes de communication

Il est possible d'établir des liens de communication entre la carte et l'extérieur (par exemple une machine hôte) de plusieurs manières.

Tout d'abord, la carte dispose d'un port série RS-232. Par défaut, c'est à travers ce lien que l'hôte communique avec la carte AP100, par exemple pour envoyer des commandes à PPCBoot. Il est également possible, toujours avec la configuration

Tableau 2.1 Différences principales entre les puces XC2VP7, XC2VP20 et XC2VP30

Modèle de puce	nb. de noyaux PowerPC	Nombre de cellules logiques	Quantité de BRAM disponible (Kb)
XC2VP7	1	11 088	792
XC2VP20	2	20 880	1584
XC2VP30	2	30 816	2448

par défaut, de connecter la carte sur un réseau Ethernet 10/100, en réglant quelques paramètres grâce à PPCBoot.

Comme nous l'avons vu plus haut, la carte est également équipée de connecteurs pour les émetteurs-récepteurs multigigabit et de deux interfaces Ethernet gigabit. Il est en revanche impossible de les utiliser avec la configuration par défaut. Pour en faire des liens de communication fonctionnels, il est nécessaire d'utiliser une configuration personnalisée intégrant la logique nécessaire à leur prise en charge.

2.1.4 Différences entre les modèles

Au niveau des composants, les différentes versions de la carte AP100 ne diffèrent que par le modèle de FPGA plateforme qu'elles embarquent : ainsi la version AP107 intègre-t-elle un XC2VP7, tandis que les versions AP120 et AP130 intègrent respectivement un XC2VP20 et un XC2VP30. Le tableau 2.1 résume les différences existant entre ces trois modèles de puces. On notera que la puce XC2VP7 est la seule parmi ces trois modèles à n'inclure qu'un unique noyau matériel de processeur, alors que la XC2VP20 et la XC2VP30 en incluent deux.

Au niveau du fonctionnement, les cartes AP107 et AP120 sont strictement identiques. L'AP130 en revanche présente une petite différence : avec cette dernière, on ne peut stocker dans la mémoire flash configuration que deux configurations au lieu de quatre.

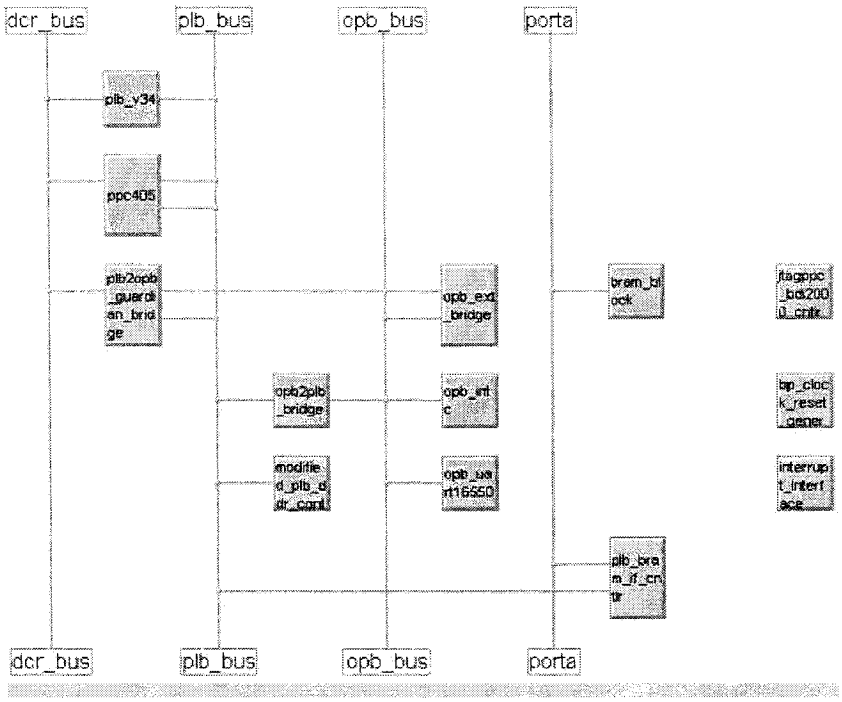

Figure 2.2 Vue du design baseline dans Xilinx Platform Studio

Maintenant que nous avons étudié brièvement ce que la carte propose, nous allons nous pencher sur ce qu'il y a à l'intérieur du composant central, i.e. le FPGA plateforme.

2.2 Design de départ

La carte AP100 est fournie avec un design de départ, baseline, préprogrammé dans la mémoire flash configuration et lancé par défaut ou en mode sans échec. Ce design, relativement simple, n'exploite pas toutes les possibilités de la carte, mais constitue un bon point de départ au développement de systèmes plus complexes. La figure 2.2 en donne la vue dans Platform Studio, l'outil principal du kit de développement EDK disponible chez Xilinx.

Les IP utilisées dans ce design rentrent dans deux catégories. Une partie d'entre

elles sont issues de la bibliothèque d'IP fournie par Xilinx avec EDK : leur code source (VHDL) est disponible et elles sont documentées. Les autres ont été conçues par Amirix : leur code source est disponible (mélange VHDL et Verilog) mais il n'existe absolument aucune documentation.

2.2.1 Processeur

L'élément central du design baseline est le processeur PowerPC, ici désigné ppc405. Comme nous l'avons vu dans le chapitre 1, les principales caractéristiques de l'IBM PowerPC 405D5 sont : architecture Harvard 32 bits, pipeline à cinq étages, deux mémoires caches (instructions et données), unité de gestion de mémoire, multiplicateur et diviseur matériels, fréquence d'horloge pouvant atteindre 400 MHz.

Le processeur exécute la partie logicielle de l'application matérielle/logicielle qu'on décide de faire rouler sur la carte. Le PowerPC 405D5 a été conçu pour utiliser l'architecture de bus IBM CoreConnect, et possède des interfaces pour PLB et DCR.

L'IP utilisée dans le design baseline est l'IP standard incluse dans la bibliothèque d'IP fournie par Xilinx.

2.2.2 Bus d'interconnexion

Les interconnexions entre les différentes IP du design baseline sont réalisées à l'aide de bus CoreConnect. L'architecture de bus IBM CoreConnect définit trois types de bus : tout d'abord, le bus PLB (Processor Local Bus) possède une importante bande passante et une faible latence et connecte le processeur et les périphériques de haute performance. Ensuite, le bus OPB (On-chip Peripheral Bus) est utilisé pour connecter les périphériques moins rapides et ainsi réduire la charge sur le bus

PLB, ce qui améliore la performance globale du système. Enfin, le bus DCR (Device Control Register) est utilisé pour permettre au processeur d'accéder directement aux registres de statut et de configuration des périphériques.

Comme le montre la figure 2.2, le design baseline possède une instance de chaque type de bus. Le bus PLB interconnecte le processeur et les contrôleurs de mémoire, alors que les autres composants sont reliés à travers le bus OPB. On note également la présence de ponts raccordant les bus PLB et OPB entre eux.

2.2.3 Périphériques

Sur le bus PLB, on trouve deux contrôleurs de mémoire. Le module *plb_bram_if_cntrl* est un contrôleur de mémoire sur puce BRAM. On trouve en conséquence, connecté directement à ce module, un bloc de mémoire, *bram_block*. Ces deux IP sont les IP standard fournies dans la bibliothèque Xilinx. Le module *modified_plb_ddr_controller* est un contrôleur de mémoire DDR SDRAM pour la mémoire externe. Cette IP est une version modifiée mais presque identique[1] de l'IP standard fournie par Xilinx.

Entre les bus PLB et OPB, on trouve deux ponts. Le module *opb2plb_bridge* est esclave sur le bus OPB et maître sur le bus PLB. Il assure que des transactions puissent être effectuées dans le sens OPB vers PLB. Il s'agit de l'IP Xilinx standard. Le module *plb2opb_guardian_bridge* est esclave sur le bus PLB et maître sur le bus OPB. Il permet les transactions dans le sens PLB vers OPB. Cette IP est une version modifiée de l'IP standard fournie par Xilinx. Le sous-module gardien ajouté par rapport à cette dernière permet d'empêcher un éventuel interblocage.

[1]Le but de cette modification reste inconnu. Pour ce qui est de sa nature, un des sous-composants est déclaré avec un port de sortie supplémentaire, mais ce port est mis en l'air (open) lors du portmap. Dans la pratique, l'utilisation de l'IP standard fonctionne très bien -du moins jusqu'à maintenant.

Sur le bus OPB, on trouve trois périphériques. Les modules *opb_intc* et *opb_uart16550* sont respectivement un contrôleur d'interruptions et un contrôleur d'UART. Ces deux IP sont les IP standard fournies dans la bibliothèque Xilinx. Le module *opb_ext_bridge* est à la fois maître et esclave sur le bus OPB. C'est un pont qui permet de réaliser des transactions, dans un sens comme dans l'autre, entre OPB et EXT, EXT étant ce qui est externe au FPGA (mémoires flash, Ethernet 10/100, etc.).

On a enfin 3 modules qui ne sont reliés à aucun bus. Ces IP ne font pas partie de la bibliothèque Xilinx. Les modules *jtagppc_bdi2000_cntlr* et *interrupt_interface* permettent respectivement d'apporter les signaux de JTAG ou les signaux d'interruption à l'intérieur du FPGA avant de les envoyer vers les entrées/sorties du processeur ppc405 ou du contrôleur d'interruptions *opb_intc*. Cette étape est nécessaire à cause du fonctionnement des outils. Le module *bp_clock_reset_generation* génère les différents signaux internes d'horloge et de remise à zéro.

Nous venons de présenter la carte AP100 et le design baseline. Ceci forme l'architecture matérielle de référence dont nous nous sommes efforcés de créer un modèle cosimulable.

CHAPITRE 3

MÉTHODOLOGIE

Nous allons d'abord expliquer comment nous avons obtenu un modèle de cosimulation de base reprenant l'essentiel des caractéristiques de l'architecture de référence, puis comment on peut modifier et compléter ce modèle de base afin de créer une grande variété de systèmes, et enfin comment il est possible de mesurer la performance des systèmes créés.

3.1 Vers un design de départ dans Seamless

Nous voulons utiliser notre modèle dans l'environnement de cosimulation Seamless CVE. Comme nous l'avons vu dans la section 1.8, Seamless fait le lien entre un simulateur logique, Modelsim, qui exécute la partie matérielle du design, et un débogueur, XRAY, qui exécute sa partie logicielle. Il faut donc s'assurer que les deux parties du design, le matériel comme le logiciel, vont être simulables dans Seamless. Or si Modelsim peut simuler n'importe quel design, chaque version d'XRAY est en revanche spécifique à un processeur. Nous avons donc commencé par tester, peu après son premier lancement, le PSP (Processor Support Package, i.e. modèle et débogueur correspondant) du Virtex-II Pro dans Seamless.

3.1.1 Test du PSP du Virtex-II Pro dans Seamless

Puisque nous voulons tester le modèle du PowerPC, il est pour l'instant inutile d'ajouter toutes sortes de modules périphériques. Nous avons donc choisi de créer

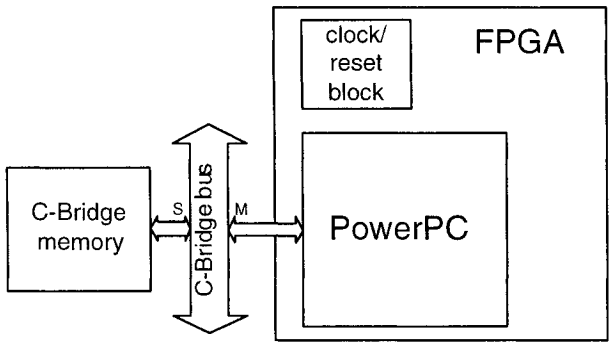

Figure 3.1 Design utilisé pour le test du PSP

un design très élémentaire, comme le montre la figure 3.1.

Si ce design est extrêmement simple, il n'a pas pour autant été facile à obtenir. En effet, le PowerPC possède un grand nombre d'entrées-sorties, et il a été nécessaire pour un certain nombre d'entre elles de piloter correctement les signaux correspondants, alors que la documentation fournie avec les premières versions était pour le moins succincte et que le modèle Seamless diffère sensiblement du modèle Xilinx ou de l'architecture de référence IBM.

Le but de cette étape étant juste de vérifier le fonctionnement basique du PSP, nous avons simplement fait rouler un morceau de code basique (une boucle avec quelques opérations arithmétiques) sur le processeur, et avons décidé de passer à la suite. Ce test a néanmoins permis de mettre en relief quelques détails intéressants. Bien évidemment, l'instanciation des composants à la main comme il a été fait ici ne constitue en aucun cas une solution viable, et nous envisagions déjà d'utiliser les outils de Xilinx pour ce faire. Néanmoins, puisque les modèles Xilinx et Seamless du PowerPC sont différents (le modèle Xilinx est un modèle RTL synthétisable standard alors que le modèle Seamless est un modèle C propre à Seamless, lié à l'ISS et supportant les fonctionnalités spécifiques de l'outil), il sera nécessaire de transformer le design obtenu avec les outils Xilinx avant de pouvoir lancer la cosimulation avec Seamless.

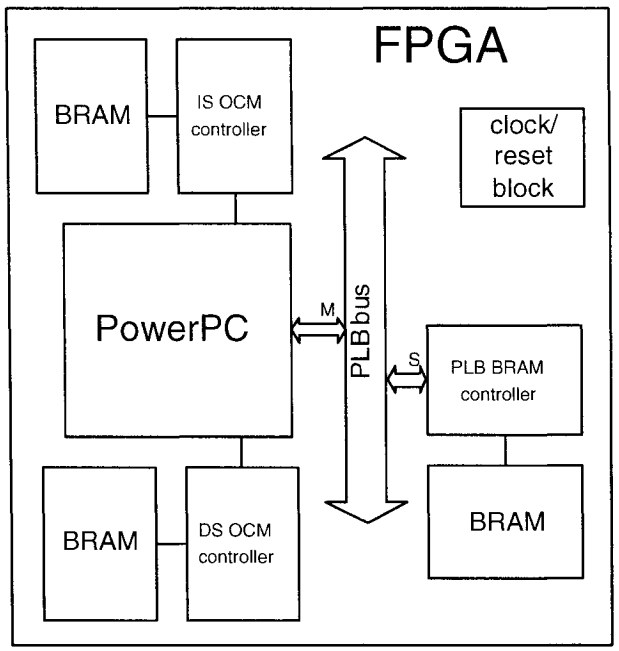

Figure 3.2 Schéma blocs du premier design créé dans XPS

3.1.2 Conception d'un design minimal dans XPS

Nous avons ensuite entrepris de créer un design simple grâce à l'outil de conception de systèmes, Xilinx Platform Studio. Un design doit au minimum contenir un processeur, de la mémoire, et un système d'interconnexions : nous avons donc instancié un PowerPC 405D5, un bloc de BRAM et son contrôleur, et un bus PLB. Cette étape était également l'occasion de se familiariser avec le PowerPC, donc nous avons décidé d'utiliser une particularité intéressante du bloc processeur présent dans le Virtex-II Pro : la présence de contrôleurs de mémoire sur puce (en anglais ISOCM et DSOCM controller pour Instruction-Side et Data-Side On-Chip Memory controller) directement interfacés sur le PowerPC. L'intérêt principal d'accéder à la mémoire à travers ces interfaces dédiées est d'assurer des accès rapides et déterministes (pas de passage à travers PLB et pas d'arbitrage). La figure 3.2 montre le schéma blocs du design obtenu.

Ce design est relativement simple : on y trouve le PowerPC, un bus PLB, deux blocs de logiques pour la gestion des horloges et de la remise à zéro, et un total de trois blocs de BRAM et leurs contrôleurs -chaque contrôleur étant connecté au PowerPC à travers le bus ou à travers les interfaces spéciales dont nous avons parlé plus haut.

Toutes les IP utilisées ici font partie de la bibliothèque Xilinx, et sont très bien documentées. XPS permet d'instancier les différents modules du design très facilement, mais il a fallu ensuite passer à travers la documentation de chacun afin de régler leurs paramètres et faire leurs connexions, ce qui représente un travail assez long la première fois et surtout très fastidieux.

On notera ici que chaque nouvelle version d'XPS a apporté son lot d'améliorations, et notamment une automatisation toujours plus importante de la phase de connexion et de réglage des paramètres des modules. Il est même désormais possible d'obtenir très rapidement un système simple grâce à l'aide d'un assistant de création.

Après que le système a été conçu dans XPS et que tous les modules ont été correctement paramétrés et connectés, l'outil est capable de générer des fichiers de simulation dans un langage HDL (nous avons choisi VHDL mais il est aussi possible d'obtenir du Verilog) en vue d'une simulation dans Modelsim. Nous avons donc maintenant la possibilité de concevoir des designs avec des outils adéquats et d'obtenir grâce à cet outil un modèle VHDL simulable dans Modelsim, le simulateur logique utilisé lors de la cosimulation dans Seamless.

Ce modèle simulable dans Modelsim n'est par contre pas nécessairement utilisable pour une cosimulation avec Seamless, même si Seamless utilise Modelsim.

3.1.3 Méthodologie de seamlessisation des designs XPS

Afin de rendre le modèle obtenu compatible pour la cosimulation dans Seamless, il faut y apporter quelques transformations. Afin de déterminer la nature des modifications à effectuer, il est nécessaire de se pencher sur le fonctionnement de Seamless.

3.1.3.1 Processeur

Tout d'abord, comme nous l'avons expliqué dans le chapitre 1, Seamless fait le lien entre un simulateur de jeu d'instructions pour le logiciel et un simulateur logique pour le matériel. Afin de permettre à Seamless de contrôler le comportement interne du processeur dans notre modèle, il faut donc remplacer le modèle Xilinx du PowerPC par son modèle Seamless, qui fait partie du PSP pour le Virtex-II Pro.

Cette étape s'annonçait très difficile. En effet, le modèle Xilinx et le modèle Seamless utilisé pour tester le PSP ne possèdent pas les mêmes entrées/sorties (pinout), ce qui présente de grosses complications. Il aurait fallu créer un adaptateur pour rendre le modèle Seamless que nous avions déjà utilisé compatible avec le modèle Xilinx.

Le modèle Seamless inclus dans le PSP est en réalité fait de plusieurs hiérarchies d'entités imbriquées, et il est possible d'employer plusieurs d'entre elles pour créer un design cosimulable. Après avoir effectué une fouille plus approfondie, il s'est avéré que le modèle utilisé pour tester le PSP est en fait un sous-composant d'un autre modèle Seamless correspondant bien cette fois au modèle Xilinx, i.e. dont les entrées/sorties sont identiques.

Il ne restait plus qu'à trouver une façon de remplacer le modèle Xilinx par le modèle Seamless correspondant. La première idée qui vient à l'esprit est d'éditer le code VHDL et de faire la substitution à la main, mais si on procède de cette manière, il faut recommencer à chaque fois qu'on modifie le design et qu'on génère de nouveaux fichiers de simulation avec XPS. Heureusement, les configurations VHDL ont ici volé à notre secours.

Le lecteur est certainement familier avec les entités, les architectures et les composants VHDL (mots clés entity, architecture et component), mais il est sans doute utile de rappeler comment fonctionnent les configurations (mot clé configuration). Les configurations permettent de spécifier quelle entité sera utilisée par chaque composant déclaré, une entité donnée étant valide si elle possède les mêmes entrées/sorties que le composant auquel on souhaite la lier.

Le PowerPC est un élément constant dans les systèmes que nous allons créer. En plaçant notre déclaration de configuration pour le PowerPC dans un fichier séparé des fichiers générés par XPS, nous pouvons donc remplacer le modèle Xilinx par le modèle Seamless dans notre simulation sans avoir à modifier aucun fichier généré.

3.1.3.2 Mémoires

Ensuite, Seamless permet des optimisations mémoire, mais afin d'utiliser ces optimisations, il est nécessaire d'utiliser des modèles de mémoire spécifiques à Seamless. Des modèles de mémoire optimisables pour la BRAM, la mémoire embarquée dans le FPGA, ont très vite été inclus dans le PSP du Virtex-II Pro. Ici encore, nous avons utilisé le principe des configurations pour spécifier quels modèles utiliser lors de la simulation.

En revanche, nous ne pouvons pas, comme dans le cas du processeur, utiliser un seul

ensemble de configurations pour tous les designs créés, car les mémoires utilisées risquent fortement de varier d'un design à l'autre. Il faudra donc en conséquence ajuster les déclarations de configuration pour les mémoires.

3.1.3.3 Touches finales

Nous avons maintenant un moyen relativement simple d'obtenir un modèle cosimulable sous Seamless du design créé avec XPS. Avant de lancer la cosimulation, il ne reste plus qu'à compiler le nouveau système modifié, et à créer la liste des stimuli.

Un des fichiers générés par XPS est un script de commandes permettant de compiler le design créé sous XPS. Même si on n'utilise pas le design tel quel et qu'on n'aurait pas besoin d'exécuter toutes ces commandes puisqu'on n'utilisera pas tous les modules XPS, on commence par exécuter ce script. Ensuite, on exécute d'autres scripts qu'on a créés afin de compiler d'une part les modèles Seamless (PowerPC et BRAM) et d'autre part le nouveau système (son nouveau top-level, i.e. la configuration).

Enfin, nous avons créé une liste de stimuli, en analysant quels signaux d'entrée devaient être pilotés pour assurer le fonctionnement de notre système.

On notera ici qu'après la mise au point de cette méthodologie, est apparu dans les versions suivantes de Seamless un assistant de transfert de designs d'XPS vers Seamless. Nous avons testé cet outil et il s'avère que, s'il pousse le degré d'automatisation un cran au-dessus (en gros, tout ce qui peut être automatisé l'est : l'assistant fait l'équivalent des scripts que nous avons créés), son utilisation n'est pas fiable à 100% (par exemple, le modèle Seamless obtenu avec l'assistant à partir du design baseline n'est pas fonctionnel).

Nous disposons donc désormais d'une méthode partiellement automatisée, mise au

point avec un design très simple, permettant de transformer un design conçu entièrement avec XPS en un modèle de cosimulation utilisable sous Seamless. Nous allons désormais essayer d'appliquer cette méthodologie au port du design de référence baseline dans Seamless.

3.1.4 Seamlessisation du design baseline

Amirix fournit le projet XPS du design baseline, mais à la différence du design minimal que nous avons utilisé précédemment, certaines IP du design baseline ne font pas partie de la bibliothèque Xilinx, mais ont été développées par Amirix.

Après avoir appliqué la méthodologie de seamlessisation à ce design, lorsque nous avons voulu lancer la cosimulation, la simulation matérielle dans Modelsim nous a posé quelques problèmes qui se sont montrés assez difficiles à régler. En fait, une fois le problème compris, la solution était simple, mais il a été assez complexe d'isoler et de comprendre la source et la nature de ces problèmes, Modelsim générant des codes d'erreur ou de mise en garde (warning) pas toujours explicites et variant avec la version de l'outil utilisée. Dépendamment de l'erreur, la simulation pouvait ne pas se lancer du tout ou bien se lancer mais ne pas fonctionner.

Au final, il s'est avéré que c'est l'IP *plb2opb_guardian_bridge* qui causait l'essentiel des problèmes, et ce pour plusieurs raisons. Par exemple, il a fallu aller modifier directement dans le code source, parce qu'ils ne correspondaient pas à ce qu'il y avait dans les primitives contenues dans les bibliothèques de simulation, le nom, le type et la valeur de génériques utilisés par des composants instanciés dans l'IP. Pourtant, le design baseline est synthétisable et fonctionne parfaitement sur la carte AP100.

Un autre problème causé par cette IP est que son code source est écrit principale-

ment en VHDL mais aussi partiellement en Verilog. Or il semblerait qu'XPS ait une faiblesse à ce niveau, et qu'il lui arrive d'oublier d'inclure certaines parties lorsqu'il génère le script de compilation pour les modules écrits à la fois en VHDL et en Verilog. Pour pallier ce problème, il a été nécessaire de créer un script additionnel qu'on exécute avant le script généré par XPS, et qui inclut les éléments manquants dans ce dernier.

On notera ici que l'utilisation de l'assistant de transfert de Seamless avec ce design résulte en l'obtention d'un modèle de cosimulation non fonctionnel.

Nous disposons désormais d'un modèle cosimulable du design baseline, et d'une méthodologie permettant d'obtenir le modèle cosimulable de n'importe quel design créé avec XPS qu'on voudrait programmer dans le Virtex-II Pro. Ce modèle reste néanmoins limité à cause de la quantité de mémoire qu'il est possible d'y inclure. En effet, chaque bloc de BRAM ne peut excéder une taille maximale de 128 ko. Puisque notre modèle ne possède pour l'instant qu'une mémoire BRAM, il permet d'utiliser au maximum 128 ko de mémoire, ce qui restreint grandement les possibilités d'applications pour la partie logicielle.

En revanche, notre modèle possède un contrôleur de mémoire externe DDR SDRAM. Nous allons donc maintenant compléter le modèle afin de permettre l'accès à la mémoire externe pendant la cosimulation dans Seamless.

3.1.5 Ajout de la mémoire externe

Il y a deux aspects important à considérer pour l'ajout de la DDR SDRAM. Le premier est que nous ne pouvons pas utiliser XPS pour le faire, car tout ce qui est conçu et instancié dans XPS va se retrouver dans le Virtex-II Pro. Or puisqu'il s'agit ici de mémoire externe, on est à l'extérieur du FPGA. Le deuxième est que si

l'on souhaite, comme avec les BRAM (mémoires internes), utiliser tout le potentiel de Seamless en cours de cosimulation, on doit ici encore utiliser un modèle de mémoire optimisable.

Puisqu'il existe une grande variété de mémoires différentes (par exemple dans notre cas, la carte AP100 intègre des puces Micron MT46V16M16TG-75), Seamless ne fournit pas de modèles de mémoires externes. Par contre, Seamless inclut l'outil PureView, qui permet de générer un modèle compatible avec les optimisations de mémoire Seamless pour un grand nombre de mémoires. Nous avons donc utilisé cet outil et généré un modèle optimisable correspondant à la mémoire qui équipe la carte AP100.

La carte AP100 intègre en fait deux puces mémoires offrant chacune une largeur de bus de données de 16 bits, pour une largeur combinée de 32 bits. L'ajout de la mémoire externe dans le modèle de cosimulation est donc un peu plus compliqué que si on n'avait eu qu'une seule mémoire de 32 bits de large, mais reste assez simple. Nous avons donc choisi de faire cet ajout à la main, en créant un nouveau fichier VHDL top-level. Il n'est pas nécessaire de modifier ce fichier tant que les entrées/sorties du modèle correspondant à ce qui est dans le FPGA ne changent pas, mais si on conçoit beaucoup de nouveaux designs avec des entrées/sorties différentes, il pourra être utile de créer un script permettant d'automatiser l'ajout de mémoire externe au design XPS.

La figure 3.3 représente notre modèle de cosimulation maintenant que la mémoire externe y a été ajoutée.

Les signaux d'entrée/sortie du modèle XPS peuvent rentrer dans une des deux catégories suivantes : ou bien ils sont en relation avec la mémoire externe, auquel cas ils sont utilisés pour connecter les mémoires au FPGA et deviennent des signaux internes, ou bien ils n'ont rien à voir avec les mémoires, auquel cas ils forment les

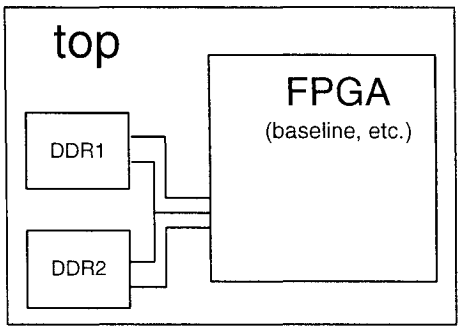

Figure 3.3 Modèle top-level avec mémoire externe

nouveaux signaux d'entrée/sortie du nouveau modèle top-level.

Maintenant que nous disposons de mémoire externe dans notre modèle de cosimulation, il est utile de s'interroger sur ce qu'on peut faire avec, mais aussi sur ce qu'on ne peut pas faire avec.

3.1.6 Récapitulatif sur les possibilités du modèle de cosimulation

Du côté partie matérielle, notre modèle de cosimulation de base possède tous les éléments du design baseline. Il a également le potentiel de posséder n'importe quel module matériel supplémentaire, l'ajout se faisant simplement en utilisant la méthodologie présentée plus haut. De plus, puisqu'il s'agit d'un modèle de simulation et pas d'un design qu'on va programmer dans le Virtex-II Pro, on ne possède aucune limitation sur la taille (nombre de portes logiques) des designs. Il est donc possible de tester en cosimulation des designs qu'il serait impossible de faire tenir sur une puce à l'instant présent mais qu'il serait possible de mettre dans des plus grosses puces plus tard.

Du côté partie logicielle, grâce à la quantité de mémoire dont le modèle dispose (64 Mo de DDR SDRAM), nous sommes assurés de pouvoir exécuter une grande gamme d'applications embarquées.

Puisque le modèle du PowerPC et les différents modèles de mémoire sont tous compatibles avec Seamless, on peut utiliser toutes les fonctionnalités de l'outil, en particulier les optimisations de mémoire, qui permettent de réduire sensiblement les durées de cosimulation.

Nous disposons désormais d'un modèle de départ cosimulable dans Seamless répondant aux critères que nous nous étions fixés. Nous allons maintenant voir comment construire d'autres systèmes à partir de ce modèle de base.

3.2 Comment faire des ajouts ou des modifications

Les modifications les plus courantes seront les modifications logicielles, i.e. celles qui concernent le code exécuté par le PowerPC.

3.2.1 Côté logiciel

Il est possible de créer des projets logiciels à partir d'XPS, avec une interface dans le genre de celle de Visual Studio. On peut donc créer, inclure et éditer des fichiers sources ou d'en-tête. Comme EDK inclut des outils de développement cross-plateforme, on peut ensuite compiler le code en un exécutable au format PowerPC. C'est cet exécutable que Seamless fait exécuter par l'ISS du PowerPC lors de la cosimulation, et il n'y aucune transformation à y apporter. Le développement d'applications logicielles utilisables avec le modèle de cosimulation n'est donc pas différent de leur développement normal.

Nous avons voulu tester une application un peu plus complexe que le test de mémoire qui servait jusqu'ici à valider le fonctionnement des modèles obtenus. Puisque les processeurs embarqués exécutent souvent des applications temps réel,

nous avons jugé opportun de tester l'exécution d'un systèmes d'exploitation temps réel (en anglais RTOS pour Real Time Operating System) dans notre modèle de cosimulation. Puisque d'une part il nous était déjà familier et d'autre part son port pour le PowerPC 405 D5 sur la carte AP100 venait d'être réalisé par des étudiants[1] de notre laboratoire, notre choix s'est porté sur uC/OS-II de la compagnie Micrium, Inc. [Micrium (2005)].

Nous avons commencé par tester une application temps réel très simple ne contenant qu'une seule tâche utilisateur. Mais même cette application extrêmement simple ne s'exécutait pas correctement. Tout démarrait normalement (initialisation du RTOS, création de la tâche utilisateur, démarrage de l'OS, exécution de la tâche utilisateur jusqu'à ce que le délai soit atteint, réordonnancement et exécution de la tâche inactive), mais l'application se retrouvait bloquée dans la tâche inactive, dont le code était alors exécuté sans fin par le processeur.

Il est rapidement devenu clair, en utilisant les capacités de débogage de XRAY (et notamment l'accès direct aux registres du processeur, qui s'est montré très utile), que le problème venait de la minuterie à intervalle fixe (en anglais FIT pour Fixed Interval Timer) du PowerPC : en effet, aucune interruption n'était générée par cette minuterie. En conséquence, la sous routine d'interruption associée n'était jamais exécutée, et donc la tâche utilisateur n'avait aucune opportunité d'être réexécutée.

Nous avons alors modifié le code du port de uC/OS-II afin d'utiliser la minuterie à intervalle programmable (PIT, de l'anglais Programmable Interval Timer) du PowerPC, mais les problèmes ont persisté. Il a été plus tard confirmé par la compagnie Mentor Graphics qu'aucune des minuteries du PowerPC n'était alors correctement supportée dans Seamless.

Il a ensuite fallu attendre, longtemps après, que le support de Mentor nous produise

[1] Éric Blanchard et Jérôme Chevalier

en avance la nouvelle version de Seamless, pour voir le FIT générer des interruptions et pouvoir exécuter notre application temps réel avec succès, période pendant laquelle la compagnie nous a quand même fait parvenir un correctif, avec lequel nous n'avons malheureusement pas eu de succès.

Puisqu'on est désormais capable de faire rouler un RTOS dans notre modèle cosimulable, on peut présumer qu'il est possible d'exécuter n'importe quelle application. On a par ailleurs expliqué que le développement de ces applications ne demande aucune attention particulière. Du côté logiciel, c'est donc très simple.

Nous allons maintenant voir comment ça se passe du côté matériel.

3.2.2 Côté matériel

Il y a plusieurs façons d'ajouter un module matériel à notre modèle de cosimulation. Nous en avons exploré trois.

3.2.2.1 IPIF

Les versions récentes d'XPS intègrent un assistant de création ou d'importation d'IP, fournissant un moyen simple d'ajouter de nouvelles IP utilisables dans l'outil.

Assistant d'importation

Dans le cas où on dispose déjà d'un module matériel dont la structure est proche de la structure standard d'un module Xilinx, l'assistant d'importation permet d'inclure ce module dans la liste des IP instanciables dans les designs XPS. L'utilisateur guide l'assistant en lui donnant le maximum d'informations utiles sur le module, et ce dernier transforme le module en une IP respectant les règles et les conventions du format Xilinx, créant au passage les fichiers manquants et nécessaires à XPS.

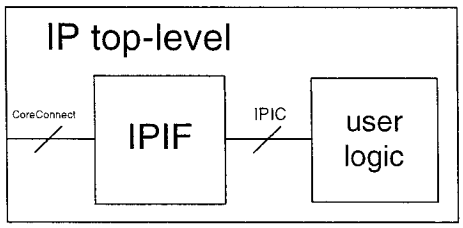

Figure 3.4 Structure d'un patron d'IP utilisant IPIF

Assistant de création

Dans le cas opposé où l'on souhaite concevoir un module à partir de rien, l'assistant de création permet de créer un patron (en anglais template) pour un nouveau module matériel. La structure du patron créé est représentée par la figure 3.4. Il apparaît clairement sur cette figure que le module est constitué de deux composants : le composant interface IP (IPIF pour l'anglais IP Interface), qui est une IP Xilinx, et le composant logique utilisateur, qui est la portion de code que l'utilisateur devra modifier afin de donner au module la fonctionnalité voulue.

Afin de créer un patron adapté à ce qu'on veut faire avec le module, l'assistant de création d'IP propose non seulement de choisir le type de bus CoreConnect auquel le module sera rattaché (PLB ou OPB), mais aussi d'inclure ou non plusieurs services IPIF comme le support d'interruptions ou le fait d'être maître sur le bus. Ces choix influencent non seulement la taille du composant IPIF, mais aussi la quantité et la variété des signaux d'interconnexion d'IP (IPIC pour l'anglais IP Interconnect) auxquels la logique utilisateur peut accéder.

L'avantage d'utiliser l'assistant de création d'IP est que tout l'aspect communication est grandement simplifié puisqu'il est géré au niveau du composant IPIF. Il est donc possible de créer des modules s'interconnectant sur les bus OPB et PLB même sans maîtriser les protocoles CoreConnect, à condition de comprendre comment fonctionne IPIF et comment utiliser les signaux IPIC.

Autre méthode

Afin de simplifier davantage la tâche de concevoir des IP Xilinx, il est aussi possible de modifier un module utilisant IPIF existant déjà et d'utiliser l'assistant d'importation sur cette IP modifiée. Xilinx inclut dans sa bibliothèque d'IP plusieurs modules génériques, i.e. sans fonctionnalité particulière, utilisant IPIF avec des configurations de services différentes, et parmi lesquels il s'agit alors de trouver celui qui possède la configuration de services IPIF la plus appropriée à la future nouvelle IP.

C'est cette méthode que nous avons utilisée au début pour créer des IP dans XPS, car c'est celle qui demandait le moins de modifications ou d'ajouts (par rapport au cas où l'on crée un patron avec l'assistant de création) à la logique utilisateur. Cependant, avec les versions plus récentes d'XPS, les patrons créés avec l'assistant de création d'IP incluent eux aussi des exemples de code fonctionnels, et donc il est devenu plus simple d'utiliser seulement l'assistant de création. De plus, l'utilisation de l'assistant de création permet plus de flexibilité, puisque dans l'autre cas il faut choisir l'IP de départ parmi un nombre de configurations de services limité.

Quelle que soit la méthode utilisée ici, il s'agit de modules décrits en langage HDL, et nous avons ensuite cherché à ajouter à notre modèle de cosimulation des modèles décrits à des niveaux d'abstraction plus élevés.

3.2.2.2 C-Bridge niveau bus

Comme nous l'avons mentionné dans la sous-section 1.8.3, Seamless inclut C-Bridge, une interface permettant de simuler et d'interconnecter avec le reste du modèle des modèles matériels écrits en C ou en C++. Les modules C-Bridge peuvent communiquer à deux niveaux distincts : bus ou pins. Puisque nous cherchons à

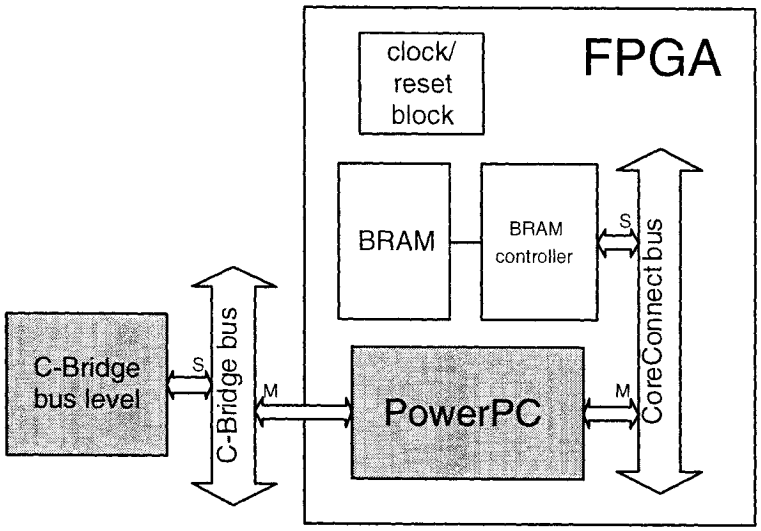

Figure 3.5 Cosimulation avec un module C-Bridge communiquant au niveau bus

augmenter le niveau d'abstraction des modules conçus, nous utiliserons ici la communication au niveau bus.

Dans ce mode de communication, le module matériel est connecté au PowerPC à travers un bus transactionnel, comme le montre la figure 3.5. Afin d'être compatible avec C-Bridge, le modèle, écrit en langage C, doit utiliser un ensemble de fonctions spécifiques faisant partie de l'API (de l'anglais Application Programming Interface) de C-Bridge, en particulier pour effectuer des transactions sur le bus C-Bridge.

Contrairement aux modules créés en utilisant IPIF, les modules C-Bridge communiquant au niveau bus ne font pas partie de la bibliothèque XPS et sont instanciés directement à partir de Seamless avant de lancer la cosimulation. Ils constituent des modules à haut niveau d'abstraction qu'il est possible de tester plus tôt dans le cycle de conception, mais ils ne sont pas connectés aux bus sur puce dans le FPGA et il est donc difficile de prédire quelle influence ils pourront avoir sur leur encombrement une fois le modèle HDL mis au point.

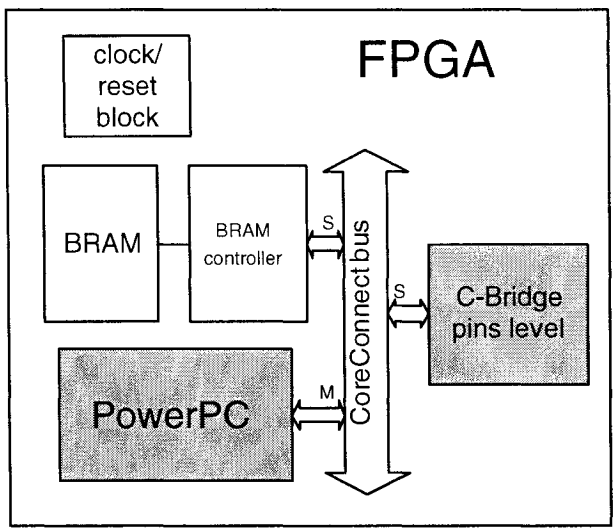

Figure 3.6 Cosimulation avec un module C-Bridge communiquant au niveau pins

3.2.2.3 C-Bridge niveau pins

C-Bridge permet un deuxième niveau de communication, au niveau pins. Dans ce mode de communication, le module C-Bridge possède un adaptateur HDL qui doit être instancié dans le modèle de cosimulation, et qui connecte les signaux d'entrée/sortie du module à n'importe quels autres signaux du modèle. La figure 3.6 montre comment un module C-Bridge communiquant au niveau pins s'intègre dans la cosimulation du modèle. Les fonctions de l'API C-Bridge utilisées pour des modules communiquant au niveau pins permettent de piloter des signaux pin par pin.

En utilisant ce niveau de communication dans C-Bridge, on dispose donc d'une solution médiane présentant et les avantages des modules créés en utilisant IPIF et les avantages des modules créés avec en utilisant C-Bridge niveau bus. D'une part les communications sont aussi détaillées que lorsqu'on utilise un module XPS et, puisqu'on peut connecter le module directement sur un bus CoreConnect, on peut mesurer l'influence du module sur l'encombrement du bus. D'autre part on peut

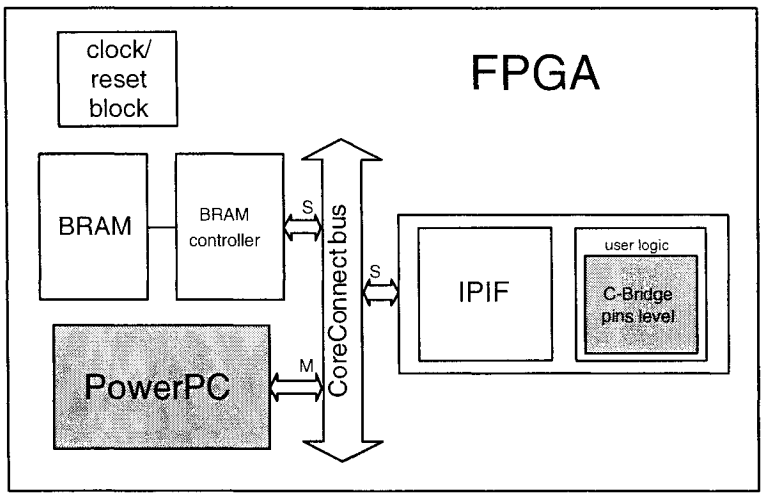

Figure 3.7 Principe de fonctionnement d'un module IPIF+C-Bridge

décrire la partie fonctionnelle du module dans un langage de programmation haut niveau.

Il reste cependant un gros obstacle à surmonter si on veut connecter nos modules C-Bridge directement aux bus CoreConnect : il est nécessaire de concevoir des adaptateurs PLB et/ou OPB pour nos modules C-Bridge niveau pins, ce qui représente un travail important.

L'idée que nous avons eue ici est de combiner IPIF et C-Bridge pour faire un module C-Bridge instanciable directement à partir d'XPS. Pour ce faire, on crée une nouvelle IP dans XPS grâce à l'assistant et, dans la logique utilisateur, on instancie l'adaptateur HDL d'un modèle C-Bridge. Il est important de bien choisir l'ensemble de signaux IPIC qu'on relaie vers le modèle C-Bridge afin de pouvoir considérer l'adaptateur comme étant plus ou moins générique. De cette manière, on peut en pratique changer le modèle C-Bridge au démarrage de la cosimulation, directement à partir de Seamless, sans avoir à repasser par XPS et générer à nouveau le modèle de cosimulation au complet. La figure 3.7 illustre ce principe de fonctionnement.

Pour les ajouts côté matériel, on a donc la possibilité d'employer des modèles à plu-

sieurs niveaux d'abstraction : avec C-Bridge et la communication au niveau bus, on a le degré d'abstraction le plus haut, dans lequel la fonctionnalité est décrite en langage C ou C++ et les communications s'effectuent au niveau transactionnel ; avec IPIF seul, on a le degré d'abstraction le plus bas, dans lequel la fonctionnalité est décrite en langage VHDL ou Verilog et les communications s'effectuent au niveau pins ; enfin, en combinant C-Bridge et IPIF, on obtient un degré d'abstraction intermédiaire dans lequel la fonctionnalité est décrite en langage C ou C++ mais les communications s'effectuent au niveau pins.

Ces trois niveaux d'abstraction permettent de définir une méthodologie de conception par raffinement progressif applicable à n'importe quel module matériel que l'on souhaite intégrer à notre modèle de cosimulation de base.

On notera ici que Modelsim permet de simuler des modèles matériels écrits en langage C ou C++, mais nous avons préféré nous concentrer sur les possibilités spécifiques à XPS et Seamless.

Maintenant que nous avons traité de l'ajout de modules logiciels ou matériels, nous allons identifier les moyens dont nous disposons pour vérifier la fonctionnalité et mesurer les performances de notre modèle en cours de cosimulation.

3.3 Vérification et performance

Seamless est un outil de covérification, et intègre donc par défaut de nombreux moyens de vérifier le bon fonctionnement du système en cours de cosimulation.

3.3.1 Vérification du fonctionnement

Du côté logiciel, l'ISS et débogueur XRAY permet de placer des points d'interruption n'importe où dans le code, d'exécuter l'application ligne par ligne (assembleur ou C), de voir et modifier la valeur de n'importe quelle variable et n'importe quel registre du processeur, et d'accéder à la mémoire. Pour ce qui est du logiciel, la visibilité et le contrôle sont donc complets.

Du côté matériel, pour les modèles HDL, le simulateur logique Modelsim permet de représenter et comparer la forme des signaux, et d'utiliser des assertions. Quant aux modèles C-Bridge, on les débogue avec DDD (Data Display Debugger), une interface graphique pour le débogueur GNU GDB (GNU Project Debugger). On peut alors placer des points d'interruptions, exécuter ligne par ligne, ou voir et modifier les variables. La visibilité côté matériel est donc complète, mais selon qu'il s'agit d'un modèle HDL ou d'un modèle C-Bridge, le contrôle qu'on a en cours de cosimulation varie grandement.

Seamless s'assure de synchroniser les simulateurs matériel et logiciel, et on peut donc arrêter et redémarrer la cosimulation afin de déboguer n'importe quel modèle.

3.3.2 Performance de la cosimulation

Il est essentiel, afin d'augmenter l'intérêt d'utiliser notre méthodologie, de s'assurer que la durée de simulation est aussi courte que possible. La vitesse de cosimulation dans un outil de cosimulation est initialement limitée par le plus lent des simulateurs, i.e. dans notre cas le simulateur logique. Nous disposons néanmoins de deux moyens d'accélérer la cosimulation.

Tout d'abord, comme nous l'avons mentionné dans le chapitre 1, Seamless permet

d'optimiser les accès mémoires (suppression des cycles matériels) de deux façons différentes : pour les lectures d'instructions, et pour les accès aux données. Pour cela, les modèles de mémoire du modèle cosimulé doivent obligatoirement être optimisables, et c'est pourquoi nous nous sommes efforcés d'utiliser de tels modèles dans notre méthodologie.

Seamless possède également un troisième mode d'optimisation, dans lequel la quasi-totalité des cycles matériels sont supprimés, et la cosimulation roule alors à la vitesse du simulateur logiciel. Bien sûr, ce mode d'optimisation n'est pas utilisable pour toutes les applications, mais il est utile de noter qu'il est possible de changer le mode d'optimisation en cours de cosimulation, et qu'on peut choisir d'activer ou de désactiver un type d'optimisation donné lors de l'exécution de telle ou telle portion de code.

Modelsim quant à lui permet d'analyser l'utilisation processeur et l'utilisation mémoire requises par la simulation d'un module en particulier. Cela peut être utilisé afin d'identifier quels modules matériels (parmi ceux créés par l'utilisateur, puisque ceux de la bibliothèque de simulation Xilinx sont certainement déjà optimisés au maximum) peuvent être modifiés en vue d'améliorer leur temps de simulation.

3.3.3 Performance du modèle

Seamless est capable de collecter des informations sur les performances du modèle en cours de cosimulation. À n'importe quel moment, il est possible d'utiliser un outil fourni avec Seamless, Seamless Performance Profiler (SPP), permettant d'analyser les informations recueillies et de les présenter sous forme de diagrammes.

SPP est capable de produire cinq types de graphiques : profileur de code, transactions mémoire, diagramme de Gantt, charge du bus et délai d'arbitrage du bus.

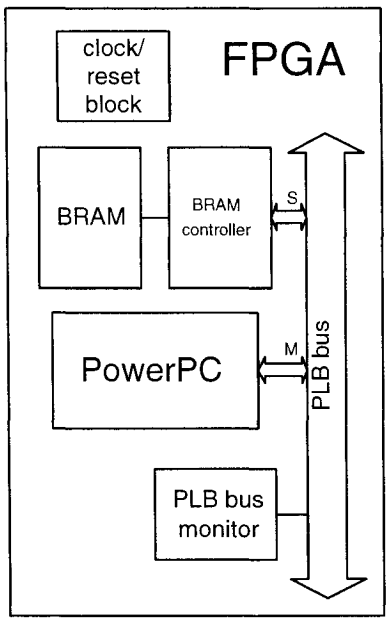

Figure 3.8 Instanciation du moniteur de bus dans le modèle de cosimulation

Il est possible d'obtenir les trois premiers sans effectuer aucune modification au modèle cosimulé. En revanche, si on souhaite capturer les informations concernant le bus, il faut modifier le modèle de cosimulation afin d'y instancier un moniteur de bus Seamless. Puisque le moniteur de bus doit accéder aux signaux du bus, il sera nécessaire de l'inclure dans le fichier généré par XPS, et donc de modifier ce fichier. Une autre solution serait d'ajouter dans XPS une IP correspondant au moniteur de bus Seamless. La figure 3.8 illustre comment le moniteur de bus s'intègre dans le modèle de cosimulation.

Les informations ainsi collectées peuvent permettre d'identifier une tâche trop gourmande en temps processeur ou un manque de bande passante mémoire, et guider le concepteur vers un autre choix de partitionnement ou d'architecture.

Dans ce chapitre, nous avons établi une méthodologie partiellement automatisée permettant d'obtenir un modèle cosimulable dans Seamless de n'importe quel système conçu avec XPS et que nous avons appliquée au design baseline. Nous avons

également mis au point une méthodologie relativement simple permettant, dans le même environnement, d'ajouter des modules matériels au modèle de base par raffinement progressif. Enfin, nous avons identifié les moyens de vérifier le bon fonctionnement d'un système et de mesurer ses performances.

Nous allons maintenant appliquer et démontrer ces méthodologies en réalisant un exemple d'application.

CHAPITRE 4

APPLICATION, RÉSULTATS ET DISCUSSION

Tout d'abord, nous allons décrire l'exemple d'application retenu. Nous appliquerons ensuite la méthodologie à cet exemple, puis, à la lumière des mesures réalisées, nous analyserons et discuterons les résultats obtenus.

4.1 Exemple d'application

Plusieurs critères influencent le choix de l'application. Avant tout, il faut pouvoir faire un partitionnement matériel/logiciel qui présente un avantage par rapport à une implémentation purement logicielle, i.e. qui puisse apporter une accélération globale. Le décodage de video MPEG4 a été envisagé au début, mais ce dernier est beaucoup trop complexe et, puisqu'il s'agit ici de démontrer une méthodologie, le choix s'est finalement porté sur la détection de contours dans des images.

Le code C++ qui a servi de point de départ a été écrit par un étudiant dont l'identité reste vague pour un de ses cours, et implémente différents calculs de détection de contours sur des images en niveaux de gris. Parmi ces méthodes de calcul, l'utilisation de l'opérateur de Sobel avec des masques de 3x3, dont suit une brève explication du principe, a été retenue. Se référer à [Fisher *et al.* (2003)] pour davantage d'information sur l'opérateur de Sobel.

L'opérateur de Sobel fait une mesure du gradient spatial. Les zones correspondant à des contours seront donc représentées par des valeurs élevées. Cette mesure est faite en apposant des masques de convolution pour chaque pixel de l'image. Ces masques,

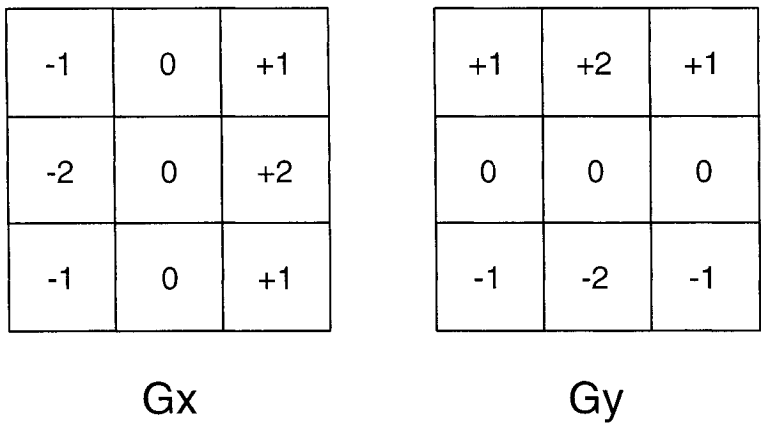

Figure 4.1 Masques de convolution de Sobel

apparaissant sur la figure 4.1, permettent de mesurer les gradients horizontal et vertical en un point de l'image.

Ces deux valeurs sont ensuite combinées en faisant leur moyenne quadratique (à une constante multiplicative près, mais on normalise ensuite), comme le montre la formule 4.1 :

$$g = \sqrt{gh^2 + gv^2} \tag{4.1}$$

où gh, gv sont les résultats des convolutions avec les masques Gx, Gy.

Il est intéressant de remarquer qu'il est donc possible, comme pour Gy dans le code, d'inverser tous les signes d'un masque sans affecter le résultat final. Afin de créer l'image de sortie, après que l'opérateur de Sobel a été appliqué à l'ensemble des pixels de l'image, les résultats sont normalisés (pour rentrer dans la plage [0..255]).

Lors de l'exécution de cet algorithme, un index se déplace dans des tableaux en 2D représentant les pixels de l'image d'entrée (lors du calcul des gradients) ou les gradients calculés (lors de la normalisation), et donc des possibilités d'accélération

matérielle se laissent déjà entrevoir. Cette application reste en outre simple, et représente donc un bon choix d'exemple pour démontrer notre méthodologie.

4.2 Application de la méthodologie

La première étape, essentielle, est la création d'un système matériel/logiciel servant de point de départ, et sa transformation en un modèle de cosimulation.

4.2.1 Obtention d'un modèle de base cosimulable dans Seamless

4.2.1.1 Matériel

En dehors du noyau de base de toute plateforme (processeur, bus, mémoire), rien d'autre n'est nécessaire pour l'application visée. Et pour ce qui est de la mémoire, seront inclus au minimum un bloc de mémoire sur puce BRAM, et au besoin de la mémoire externe DDR SDRAM. La figure 4.2 est une représentation de tout ce qui peut potentiellement former la plateforme finale.

Il est important de garder à l'esprit que la complexité de la plateforme conçue et la taille de ses éléments influencent directement la rapidité de la simulation. En effet, plus elle intègre d'IP et plus ces IP sont de taille importante, plus le simulateur logique a de travail. Or c'est justement lui qui limite la vitesse de la cosimulation. En conséquence, il est préférable de partir d'un design aussi simple que possible. Il sera de toute façon possible de le compléter à n'importe quel moment en fonction des besoins ressentis. Le design de départ pour cette application, représenté sur la figure 4.2 entre les deux lignes pointillées, a été conçu dans XPS uniquement à l'aide d'IP faisant partie de la bibliothèque d'IP Xilinx (sauf le moniteur de bus, qui est une IP Seamless et qui est inclus plus tard dans le design).

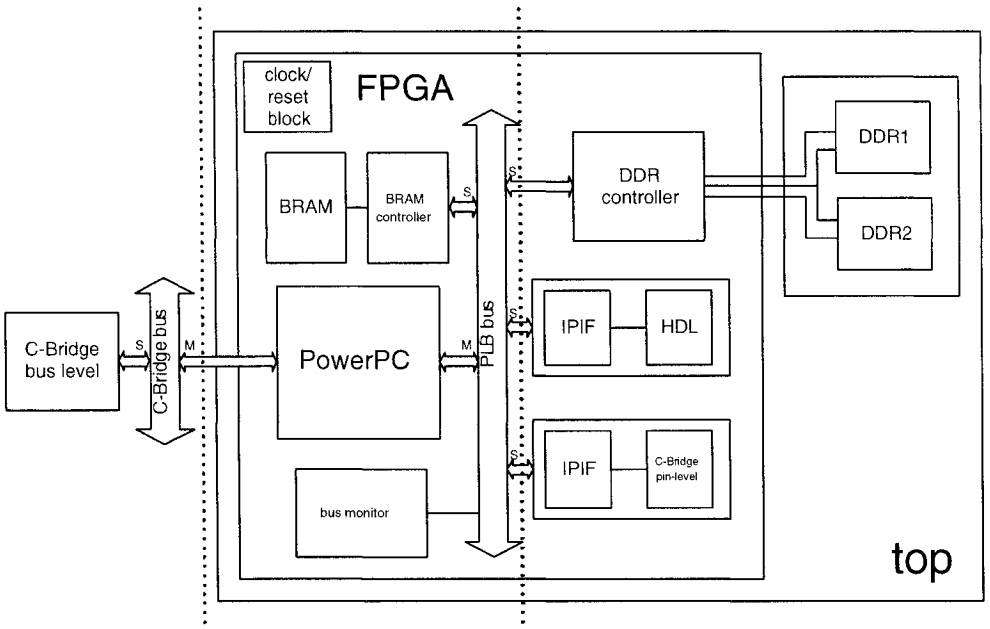

Figure 4.2 Plateforme finale

4.2.1.2 Logiciel

Plusieurs modifications ont été apportées au code récupéré avant d'en faire le code de départ qui sera exécuté par le processeur embarqué dans la plateforme de départ. Tout d'abord, il a été modifié pour être du C. Ensuite, tout l'aspect traitement de fichiers a été supprimé : à la place, les données (la valeur de gris de chaque pixel) de l'image d'entrée et de l'image de sortie théorique sont stockées dans deux tableaux de constantes. Après avoir effectué la normalisation, plutôt que de les écrire dans un fichier, le programme vérifie que les résultats qu'il a calculés correspondent aux résultats théoriques. Le tableau 4.1 illustre, à l'aide de pseudo-code, les différences de fonctionnement entre le code récupéré et le code de départ.

Tableau 4.1 Différences entre code récupéré et code de départ

Code récupéré	Code de départ
lire fichierIN vers tabImageIN	inclure tabImageIN
calculer résultats (Sobel&autres)	inclure tabImageOUTThéorie
écrire résultats vers fichierOUT	calculer résultats (Sobel&autres)
	vérifier résultats = tabImageOUTThéorie

4.2.1.3 Seamlessisation

Côté matériel, les fichiers de simulation pour la plateforme de départ sont générés à l'aide d'XPS, puis le système au complet est compilé. Il faut alors créer le fichier de configurations pour le système afin de remplacer les modèles Xilinx du PowerPC et des BRAM par leurs modèles Seamless, puis compiler ces modèles Seamless et les configurations.

Côté logiciel, il suffit de cross-compiler le code de départ à partir d'XPS.

Pour terminer, il ne reste qu'à créer les stimuli permettant de piloter les signaux d'entrée du système. Le design de départ est désormais prêt à être cosimulé dans Seamless.

4.2.2 Ajout du moniteur de bus

Il faut instancier et connecter le moniteur de bus dans le système, ce qui implique de modifier le fichier top-level du système de départ. Ce nouveau fichier top-level et le fichier de configurations sont ensuite recompilés. Le nouveau système ainsi créé sera par la suite appelé système standard, car il sera à partir de maintenant utilisé pour toutes les cosimulations, sauf s'il devient nécessaire de le modifier afin de répondre à des besoins particuliers. Le reste (logiciel, stimuli) demeure inchangé.

Maintenant que le modèle intègre le moniteur de bus, il est possible de recueillir et

d'analyser tous les types de données avec le SPP.

Puisque le système standard sera systématiquement utilisé par la suite (sauf s'il ne répond plus aux besoins), il est intéressant de s'interroger sur ses limitations (en rapport avec l'application ciblée, i.e. la détection de contours Sobel 3x3), donc sur les raisons éventuelles de le modifier. En fait, sa principale limitation est la quantité de mémoire qu'il intègre. Même si, puisqu'il s'agit d'une simulation, la taille de la BRAM peut théoriquement avoir une valeur arbitraire, l'IP du contrôleur de BRAM ne supporte pas une taille de BRAM supérieure à 128 ko.

Le système standard ne permettrait donc pas de cosimuler des designs pour lesquels les besoins en mémoire sont supérieurs à 128 ko. Dans ce cas, il existe plusieurs solutions pour pallier le problème. Il est par exemple possible d'ajouter un contrôleur de mémoire externe DDR SDRAM comme dans la figure 4.2, mais il serait aussi envisageable d'ajouter une deuxième BRAM. Dans ce dernier cas, une des BRAM serait par exemple utilisée uniquement pour les instructions alors que l'autre serait utilisée uniquement pour les données.

4.2.3 Profilage et choix de partitionnement

Afin de bien voir, pendant le profilage du programme, quelles sont les étapes les plus gourmandes en temps CPU, le code exécuté par le PowerPC a été modifié afin que le traitement soit réalisé par plusieurs sous-fonctions ayant des noms parlant, comme le montre le pseudo-code suivant :

```
include tableauFileDataIN
include tableauFileDataOUT
main
    createDynamicArrays
```

```
sobel_operator
    convert1Dto2D
    addFrame
    compute3x3mask
    scaleToRange255
fake_print
    checkResultsAreCorrect
delete_dynamic_arrays
```

Comme le montre la figure 4.4, c'est la fonction *compute3x3mask* qui consomme le plus de temps CPU. C'est donc cette fonction qui va être transformée en un module matériel dans le but d'accélérer l'exécution globale de l'application.

4.2.4 Partitionnement pseudo-matériel/logiciel avec uC/OS-II

Le système standard est ici utilisé comme plateforme matérielle. Le code exécuté par le PowerPC, en revanche, est lourdement modifié afin d'apporter les changements suivants.

Pour commencer, le PowerPC roule désormais le RTOS uC/OS-II. Dans cet environnement multi-tâches, il est possible de créer plusieurs tâches utilisateur. Par exemple, une ou plusieurs de ces tâches peuvent servir à représenter ce qu'exécute le processeur, auquel cas elles seront dites tâches logicielles, tandis que d'autres peuvent servir à représenter un ou plusieurs modules matériels, auquel cas elles seront dites tâches matérielles. Bien que l'application s'exécute encore dans un environnement purement logiciel, il est donc possible de modéliser un système matériel/logiciel. Pour l'exemple de la détection de contours, il y a deux tâches utilisateur : l'une logicielle pour le processeur, l'autre matérielle pour le module matériel remplaçant le calcul logiciel de *compute3x3mask*.

Ensuite, puisque maintenant le processeur n'exécute plus l'application tout seul mais doit coopérer avec un module matériel (du moins, c'est ce qu'il faut s'imaginer ici), l'application doit être repensée. C'est aussi l'occasion de se rapprocher le plus possible d'une implémentation où il y aurait un vrai module matériel, et donc de se rapprocher du modèle noyau-interface, le noyau s'occupant de la fonctionnalité et l'interface se chargeant des communications.

Du côté fonctionnalité, les modifications sont importantes et transforment radicalement les algorithmes utilisés jusqu'ici. Tout d'abord, les tableaux en 2D ont été complètement abandonnés, ce qui marque au passage la disparition de l'étape d'ajout de cadre. Afin de minimiser à la fois le nombre d'accès au bus et l'espace mémoire utilisé, au lieu d'envoyer à chaque fois neuf pixels pour le calcul du gradient d'un pixel, on utilise dans la tâche module l'équivalent de registres à décalage. Enfin, au lieu de stocker tous les résultats avant de normaliser, les résultats sont vérifiés au fur et à mesure qu'ils arrivent à la tâche processeur.

L'aspect communications est nouveau puisqu'avant d'effectuer le partitionnement, il n'y en avait pas besoin. La communication inter-tâches est ici faite de manière à se rapprocher d'une communication à travers un bus. La figure 4.3 illustre comment l'utilisation de deux boîtes aux lettres pour faire communiquer les deux tâches utilisateur permet de simuler assez efficacement un accès (atomique) de type poignée de main entre un processeur et un module esclave à travers un bus transactionnel. Ici, la lecture d'une boîte aux lettres est bloquante, alors que l'écriture dans une boîte aux lettres ne l'est pas. Les deux boîtes aux lettres sont vides au début de l'accès, et la tâche modélisant le matériel (esclave) est la plus prioritaire.

Un autre type de communications inter-tâches a également été implémenté. Il s'agit d'une sorte de rendez-vous bilatéral effectué avec des boîtes aux lettres, minimisant ainsi le nombre d'accès à ces boîtes aux lettres au lieu de copier un mode de

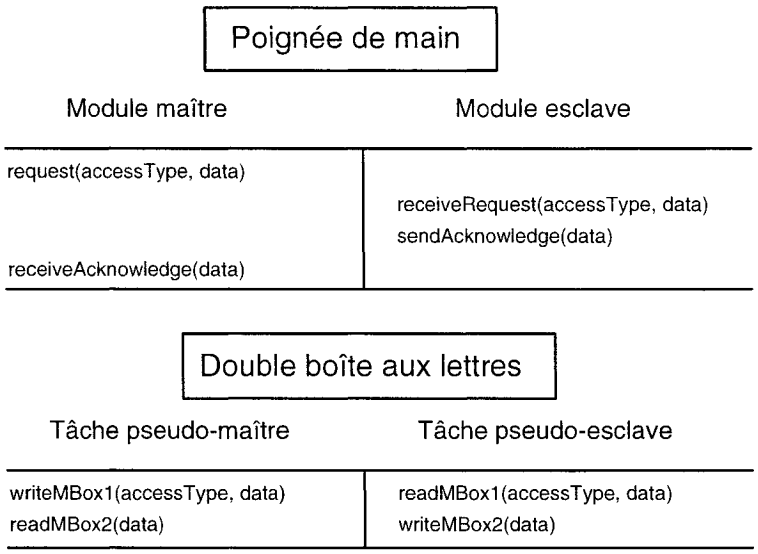

Figure 4.3 Simulation d'une poignée de main avec des boîtes aux lettres

communication par poignée de main.

4.2.5 Partitionnement matériel/logiciel avec C-Bridge niveau bus

Le système standard est ici encore utilisé comme plateforme matérielle.

Pour la première fois, le processeur n'exécute pas toute l'application mais est libéré d'une partie du traitement, i.e. du calcul de gradient avec l'opérateur de Sobel. En prenant comme référence l'étape précédente avec les deux tâches, le PowerPC exécute désormais le code qu'exécutait la tâche logicielle, tandis que le code qu'exécutait la tâche matérielle devient la responsabilité du nouveau module matériel. En outre, puisque le processeur n'exécute plus qu'une seule tâche, on abandonne le RTOS.

Du côté logiciel, il est donc extrêmement simple de passer de l'étape précédente à cette étape. La seule différence qui existe entre le code qu'exécute maintenant le PowerPC et celui qu'exécutait la tâche logicielle est la façon dont processeur

et module communiquent. Il suffit désormais d'accéder aux registres du module matériel à l'aide de pointeurs.

Le nouveau module matériel est quant à lui un modèle C-Bridge communiquant avec le PowerPC au niveau bus, donc de manière complètement indépendante du reste du matériel (le système standard). Il reprend la fonctionnalité de la tâche matérielle de l'étape précédente, dans sa forme de code C, mais son mode de communication est très différent. L'essentiel du travail à réaliser pour ce modèle est donc de créer, en utilisant l'API de C-Bridge, une interface de communication pour un module esclave.

4.2.6 Partitionnement matériel/logiciel avec IPIF et C-Bridge niveau pins

Le système standard ne suffit plus ici, et il faut l'adapter en lui ajoutant le module IPIF+C-Bridge.

Lors de l'étape précédente, le module matériel communiquait avec le processeur à travers un bus transactionnel séparé de la plateforme matérielle, le bus C-Bridge. Maintenant que le module matériel IPIF+C-Bridge fait partie de la plateforme matérielle, le PowerPC y accède à travers le bus PLB. Du côté logiciel, ceci n'a aucune influence, et le code exécuté par le PowerPC est le même que celui qu'il exécutait à l'étape précédente.

C'est du côté matériel que les changements se font sentir. Tout d'abord, si lors de l'étape précédente, à la fois le noyau fonctionnel et l'interface de communication du module sont tous deux écrits en C, la tâche est ici séparée entre une partie C qui implémente le noyau et une partie VHDL qui implémente les communications. Cette hétérogénéité implique en fait un niveau de communication supplémentaire,

inutile dans le cas précédent du module C-Bridge communiquant au niveau bus, entre les deux parties du module.

La partie C-Bridge n'est donc en réalité pas complètement dégagée de la nécessité de communiquer, mais au lieu de le faire directement avec le bus, elle communique avec le module IPIF, qui joue donc un rôle d'adaptateur vers CoreConnect PLB. En conséquence, le module C-Bridge précédent (communiquant au niveau bus) est modifié afin de changer sa façon de communiquer. L'aspect fonctionnalité du modèle reste en revanche identique.

Cette étape constitue clairement un raffinement des communications entre le processeur et le module matériel.

4.2.7 Partitionnement matériel/logiciel avec IPIF et VHDL

La seule différence existant entre cette étape et la précédente est que le noyau est ici implémenté en VHDL et pas en C. Le passage de l'une à l'autre n'est pas pour autant immédiat. Pour l'application visée ici, la quantité et la complexité du code à produire sont très acceptables. De plus, l'algorithme de calcul des modules matériels écrits en C reflète l'implémentation HDL avec des registres à décalages. Pour d'autres applications en revanche, ce sera probablement une étape laborieuse.

Cette étape constitue un raffinement de la partie fonctionnalité du module.

4.2.8 Récapitulatif

Le tableau 4.2 récapitule les propriétés principales des différents modèles matériels créés lors de l'application de la méthodologie à l'application choisie.

Il est utile de noter qu'afin de faciliter la comparaison des modèles entre eux, il est

Tableau 4.2 Récapitulatif des propriétés des modèles matériels

Modèle	Matériel-interface	Matériel-noyau	Communication avec le processeur
2 tâches uC*	C	C	pseudo-bus transactionnel
C-Bridge bus	C	C	bus transactionnel
IPIF+C-Bridge pins	VHDL (&C)	C	bus pin-accurate
IPIF	VHDL	VHDL	bus pin-accurate

*Dans le cas où deux tâches uC/OS-II s'exécutent sur le processeur, le module matériel n'est en fait pas vraiment un module matériel.

tout à fait possible d'utiliser une combinaison des différents modules matériels en même temps dans un système. Il suffit pour cela de donner des adresses de base différentes à chacun d'eux.

4.3 Mesures

4.3.1 Profilage de l'application

Le profilage de l'application purement logicielle est réalisé en utilisant le SPP. Comme le montre la figure 4.4, c'est l'étape de calcul de gradient qui consomme le plus de ressources. C'est la raison pour laquelle c'est cette partie du programme qui a été déplacée en matériel.

4.3.2 Durée des cosimulations

La durée des cosimulations est un élément très important. En effet, pour que la méthodologie apporte un réel gain de temps de conception, il faut s'assurer que les durées de simulation sont aussi petites que possibles.

Le tableau 4.3 indique la durée de cosimulation de chaque implémentation réalisée

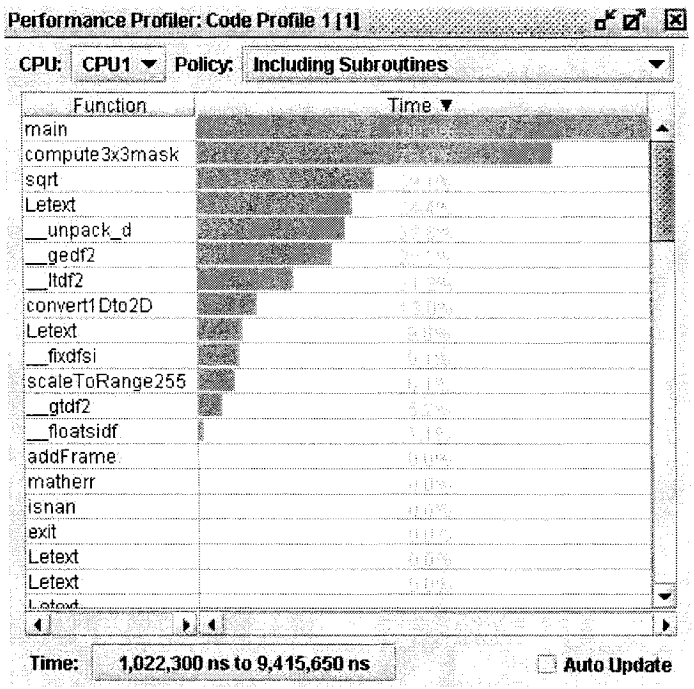

Figure 4.4 Profilage de l'application logicielle avec SPP

lors de l'application de la méthodologie à l'exemple de la détection de contours avec l'opérateur de Sobel. Ces mesures sont faites à l'aide d'un chronomètre. Si la durée est raisonnablement longue, il est démarré lorsque l'exécution du programme logiciel est lancée, et arrêté lorsque le programme a obtenu et vérifié les résultats pour l'image au complet. Sinon (durée de cosimulation trop grande), il est arrêté à un temps donné suffisamment grand, et la durée totale est calculée en regardant où le programme est rendu (indice du pixel traité) et en appliquant une règle de trois. Les conditions sont les mêmes pour toutes les mesures, effectuées en étant le seul utilisateur d'un serveur Sun Blade 100 équipé de 2 Go de RAM.

Il est important de préciser que pour toutes ces mesures, les optimisations dans Seamless ont été réglées au maximum. En conséquence, il s'agit ici du temps minimum qu'il est possible d'atteindre. Désactiver un ou plusieurs types d'optimisations risque fortement d'avoir des conséquences catastrophiques sur la durée des cosimulations.

Tableau 4.3 Durée de la cosimulation pour chacune des implémentations

Implémentation	Durée de la cosimulation
(1) purement logicielle avec l'algorithme modifié*	6 min 30 s
(2) pseudo-mat./log. uC/OS-II avec pseudo-bus	3 h 00 min 00 s
(3) pseudo-mat./log. uC/OS-II sans pseudo-bus	2 h 30 min 00 s
(4) mat./log. avec C-Bridge niveau bus	2 min 35 s
(5) mat./log. avec IPIF et C-Bridge niveau pins	28 min 00 s
(6) mat./log. avec IPIF (et HDL)	28 min 00 s

*Le code exécuté ici par le processeur est équivalent au code exécuté par les deux tâches dans l'implémentation pseudo-matérielle/logicielle avec uC/OS-II.

4.3.3 Performances diverses

Le SPP peut fournir plusieurs types d'informations sur les performances du système. Le profileur de code a déjà été utilisé afin de déterminer quelle partie de l'application transformer en un module matériel. Il est aussi possible d'obtenir un diagramme de Gantt logiciel, des informations sur les accès mémoire, ainsi que des informations sur les accès à travers le bus et l'arbitrage du bus.

Afin de pouvoir exploiter les informations disponibles, il est nécessaire de prendre certaines précautions, en paramétrant comment on observe ces données de façon appropriée. D'une manière générale, il ne faut pas trop zoomer. Par exemple, on ne peut rien tirer de la figure 4.5. En revanche, si on augmente l'intervalle de temps couvert, on obtient une mesure du taux d'utilisation du bus (environ 10% en moyenne), qui aurait pu servir à identifier un engorgement du bus et conduire à la modification de l'architecture du système.

Si on regarde du côté des transactions mémoire, comme illustré sur la figure 4.6, on peut facilement identifier les portions de code gourmandes en accès mémoire, ce qui peut amener le concepteur à réaliser des améliorations logicielles. Le SPP est également capable d'afficher des informations sur les accès à la mémoire cache, et notamment de préciser s'il s'agit de succès ou d'échecs. Cette possibilité est très

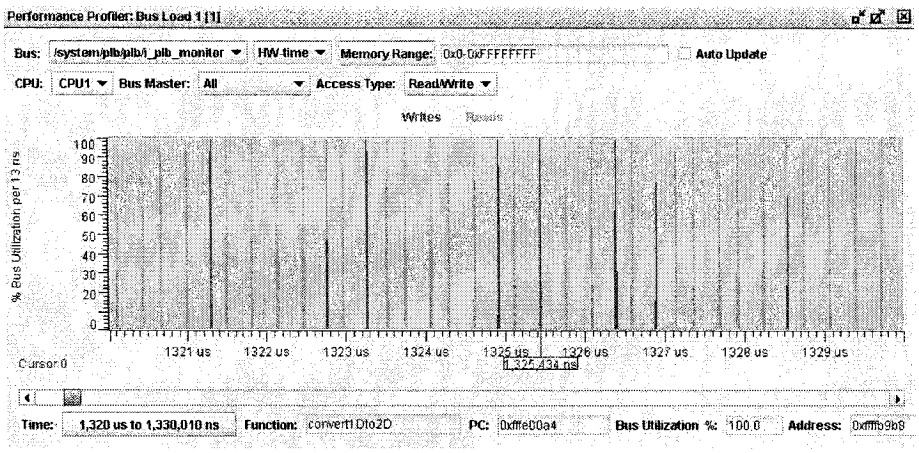

Figure 4.5 Informations sur les transactions à travers le bus PLB dans SPP

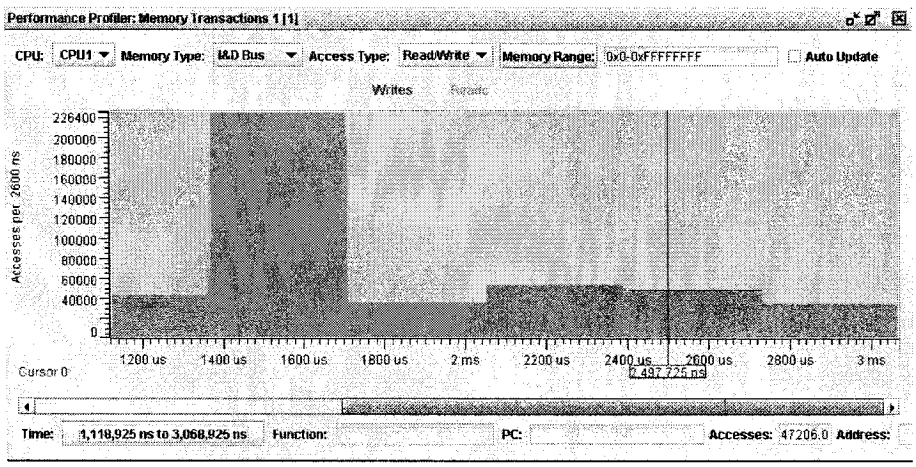

Figure 4.6 Informations sur les transactions mémoire dans SPP

utile lorsqu'on décide d'utiliser la mémoire cache du PowerPC et permet de voir facilement l'effet des différentes méthodes de gestion de la cache sur les performances de l'application.

4.4 Analyse des résultats et discussion

4.4.1 Durées de simulation

Comme le montre le tableau 4.3, les durées de simulation varient grandement d'une implémentation à l'autre. L'énorme différence existant entre la première implémentation (1) et les deux suivantes (2) et (3), alors qu'elles sont pourtant toutes les trois logicielles, s'explique par le fait que les implémentations (2) et (3) roulent deux tâches uC/OS-II. Le RTOS représente une couche logicielle beaucoup plus gourmande en temps CPU que l'application à proprement parler : il répond aux interruptions (de la minuterie), réordonnance régulièrement les tâches (ce qui d'ailleurs est ici inutile, étant donnée la façon donc les tâches sont synchronisées entre elles) et exécute régulièrement des tâches en arrière-plan.

Afin de mieux comprendre comment le RTOS peut avoir une telle influence sur la durée de cosimulation, nous avons effectué le profilage de l'application pour l'implémentation (2). La figure 4.7 présente les résultats obtenus. Il apparaît que les fonctions propres à uC/OS-II (celles qui commencent par OS) comptent pour plus de 84.3% du temps CPU total. La fonction la plus gourmande, qui utilise plus de la moitié du temps CPU, *OS_MemCopy*, est appelée par le RTOS pendant les lectures et écritures vers les boîtes aux lettres. Les fonctions *OSIntExit* et *OSIntCtxSw* quant à elles sont exécutées respectivement quand on sort d'une interruption ou qu'on réalise un changement de contexte.

La différence entre les deux implémentations uC/OS-II (2) et (3) se fait au niveau de la méthode de communication inter-tâches. Lorsqu'on cherche à modéliser une poignée de main avec deux boîtes aux lettres, il faut une lecture et une écriture vers une boîte aux lettres par tâche par accès au bus (par passage d'une donnée d'une tâche vers l'autre), alors qu'avec l'autre méthode de synchronisation (pseudo-

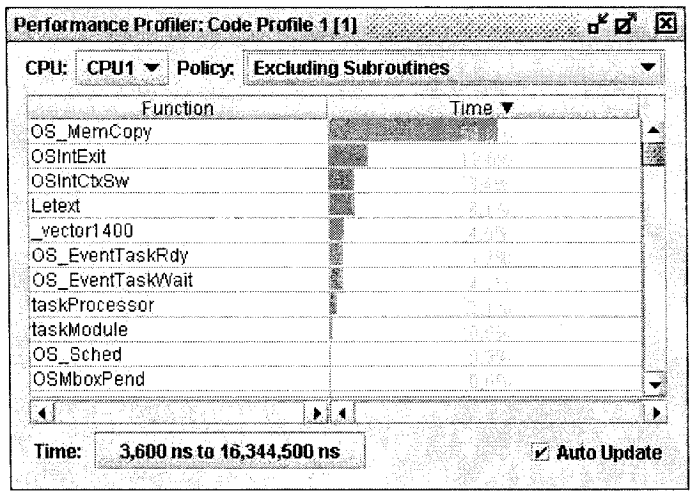

Figure 4.7 Profilage de l'application uC/OS-II avec SPP

rendez-vous bilatéral), il suffit d'une lecture ou d'une écriture vers une boîte aux lettres par tâches pour transmettre une donnée de l'une vers l'autre. L'implémentation avec poignée de main (2) génère donc environ deux fois plus de lectures et écritures vers des boîtes aux lettres que l'autre (3), ce qui explique que cette dernière s'exécute plus rapidement.

La durée de cosimulation de la première implémentation réellement matérielle (4), avec un modèle C-Bridge communiquant au niveau bus avec le processeur, est plus rapide que l'implémentation purement logicielle (1). Ceci est dû au fait que le code C du programme et le code C du modèle C-Bridge ne sont pas exécutés de la même manière. Le programme est exécuté par un ISS, donc il s'agit d'une simulation d'exécution, alors que le modèle C-Bridge est exécuté sur la machine (le serveur). Par ailleurs, le modèle C-Bridge communique avec le PowerPC à travers un bus transactionnel dédié, le bus C-Bridge. Puisque le noyau s'exécute plus rapidement et que l'interface de communication n'ajoute pas de retard important, cette implémentation est donc plus rapide à cosimuler que l'implémentation purement logicielle.

L'implémentation avec IPIF seulement (6) est plus longue à cosimuler que l'implémentation purement logicielle (1), ce qui est tout à fait attendu, puisqu'il est nécessaire de passer à travers le bus sur puce pour accéder au module IPIF, et de simuler du matériel. De plus, les registres du modèle IPIF ne peuvent pas être rendus optimisables, et donc Seamless simulera toujours tous les accès qui y sont faits. Ce qui est plus intéressant, c'est de voir que l'implémentation avec IPIF est plus rapide à cosimuler que celles avec uC/OS-II d'un facteur 5 à 6 fois environ. Ceci réaffirme à quel point un RTOS peut exécuter une grande quantité de code.

Pour l'implémentation IPIF+C-Bridge pins (5), les résultats sont identiques à ceux de l'implémentation avec IPIF seulement (6). Ceci s'explique par le fait que dans l'implémentation IPIF seul, le temps passé à la simulation du noyau HDL du module matériel représente une toute petite partie du temps de simulation matérielle global. Or cette petite fraction de temps est la seule portion que le modèle C-Bridge pins puisse accélérer.

4.4.2 Autres informations

Comme il a été dénoncé un peu plus haut, les informations données par le SPP ne permettent pas de bien mesurer les performances du système. Elles peuvent quand même fournir quelques renseignements intéressants. Par exemple, pour l'implémentation purement logicielle sans RTOS (la première de la liste), on a environ 25 fois plus d'accès en mémoire de type instructions (donc lectures) que d'accès de type données (lectures et écritures confondues). Cela montre que notre programme se comporte comme une application orientée contrôle plutôt que données. La modification de l'algorithme pour effectuer les calculs de gradients sans passer par des tableaux 2D a en effet rajouté beaucoup de contrôle dans l'application.

Toujours pour la même implémentation, on se rend compte que désactiver les op-

timisations fait chuter la fréquence des accès mémoire d'un facteur 500. Lorsque toutes les optimisations sont activées et que la mémoire dans le système est optimisable, Seamless accède à la mémoire au travers du serveur de mémoire cohérente, et la simulation matérielle des accès mémoire n'est donc même plus nécessaire. Il est aussi intéressant de voir que toute activité sur le bus cesse lorsque toutes les optimisations sont activées, car ici le processeur est le seul maître sur le bus et il n'a besoin d'accéder au bus que pour lire ou écrire en mémoire, ce qu'il ne fera même pas (accéder au bus) si les optimisations sont activées.

4.4.3 Temps de développement

Un point important qui n'a pas encore été abordé est celui du temps de développement, puisqu'il influence la viabilité d'une méthodologie. En effet, à quoi bon développer un module à haut niveau d'abstraction si il est plus rapide de le développer directement à bas niveau d'abstraction ?

Le développement d'un module C-Bridge est assez laborieux la première fois. Le noyau peut être n'importe quel code C++, mais l'interface de communication doit respecter le modèle de communication prévu par C-Bridge, que ce soit à l'un ou l'autre des deux niveaux disponibles (bus et pins). L'essentiel du travail est donc de créer une interface de communication C-Bridge plus (niveau bus) ou moins (niveau pins) générique, et d'y greffer le noyau fonctionnel voulu.

Le développement du modèle IPIF devra être réalisé un jour ou l'autre, et ne constitue donc pas une étape facultative. Là encore, il faudra d'abord se familiariser avec IPIF et les signaux IPIC, mais ensuite tout dépend de la complexité du traitement à réaliser.

Le développement de modèles C-Bridge a donc sa place dans la méthodologie. En

particulier les modèles communiquant au niveau bus, qui sont non seulement plus faciles à concevoir, mais aussi plus rapides à cosimuler. Ils seront donc l'implémentation de choix dans les premières phases du cycle de conception.

4.4.4 Seamless ou non

Le flot de conception standard EDK pour créer des systèmes sur puce programmable pour le Virtex-II Pro implique de développer directement les modules en langage HDL, les déboguer en les simulant dans un simulateur matériel, puis les synthétiser et les envoyer dans le FPGA. L'exécution sur une carte de développement est très rapide et il existe des moyens de déboguer l'application dans ces conditions (par exemple JTAG).

L'intégration de Seamless dans le flot de conception donne l'opportunité d'intégrer des modules écrits en C dans un système à n'importe quel moment et de le cosimuler. Seamless permet en outre lors du débogage un contrôle complet sur ce qui se passe dans le système. En revanche, la cosimulation sera toujours beaucoup plus lente que l'exécution à partir d'un FPGA.

Les deux solutions présentent donc des avantages et des inconvénients. Il y a pourtant plusieurs situations dans lesquelles Seamless sera très appréciable, et notamment lorsqu'il y a des problèmes difficiles à déboguer sans les capacités propres à un cosimulateur. Seamless permet également de valider une spécification fonctionnelle avant de commencer à développer le RTL.

On notera ici qu'en conséquence de changements récents du côté de C-Bridge, la version que nous avons utilisée pour ce travail ne sera bientôt plus supportée du tout dans Seamless.

CONCLUSION

Résumé du travail accompli

Nous avons dans un premier temps créé un modèle cosimulable dans Seamless du design baseline pour la carte AP100. Ce modèle supporte notamment l'ajout de mémoire DDR externe au FPGA. Dans un second temps, nous avons défini une méthodologie de codesign pour systèmes sur puce programmable adaptée au flot de conception Xilinx pour Virtex-II Pro. Cette méthodologie permet d'obtenir le modèle de cosimulation dans Seamless de n'importe quel design conçu avec XPS, et ainsi crée un pont entre les deux flots de conception XPS et Seamless. La méthodologie mise au point permet également la conception de modules matériels par raffinement progressif et permet la validation de ces modules à chaque niveau d'abstraction. Enfin, nous avons démontré l'utilisation de cette méthodologie à l'aide d'un exemple d'application. Les objectifs que nous nous étions définis ont donc été atteints. Il serait néanmoins possible de compléter ces travaux et d'explorer des idées connexes.

Travaux futurs

Tout d'abord, on pourrait étudier comment utiliser cette méthodologie en parallèle avec SPACE. Dans SPACE, tous les modules sont écrits en C++, et on peut facilement les faire passer de logiciel vers matériel et vice-versa. Les communications quant à elles sont raffinées petit à petit lorsqu'on passe d'un niveau au niveau suivant. L'utilisation de modules matériels IPIF+C-Bridge pins permettrait de raffiner les communications vers un niveau HDL tout en gardant le noyau fonctionnel des modules en C++ à haut niveau d'abstraction. Pour automatiser le plus possible le

passage du niveau précédent à ce nouveau niveau, il faudrait alors créer un module IPIF+C-Bridge pins générique utilisable pour n'importe quel module SPACE.

Il serait également très intéressant d'explorer l'idée d'un module matériel exécuté par un processeur logiciel. Puisque nous visons des Virtex-II Pro, il s'agirait de faire exécuter le noyau fonctionnel du module par un MicroBlaze. Malheureusement, il n'est pour l'instant pas possible d'inclure cette possibilité dans notre méthodologie car il n'y a pas d'ISS du MicroBlaze disponible dans Seamless. Néanmoins, si tous les modules matériels sont modélisés de cette manière, il est tout à fait possible de se passer de la cosimulation et de lancer le système directement sur le FPGA, la taille (nombre de portes logiques) de la puce limitant alors le nombre de modules qu'il est possible d'inclure dans le système.

On notera enfin que, si on souhaite continuer à utiliser des modèles C-Bridge, il sera nécessaire de les modifier afin de les rendre compatibles avec le nouveau standard. En effet, Mentor a annoncé que les prochaines versions de Seamless ne supporteraient plus du tout le standard C-Bridge que nous avons utilisé ici.

Considérations finales

Si on considère, d'une part les efforts réalisés par Mentor Graphics pour le support du Virtex-II Pro dans Seamless, avec notamment le lancement d'une version spéciale, SeamlessFPGA, et d'autre part la hausse du nombre de compagnies qui semblent utiliser cet outil, il apparaît que la conception de systèmes sur puce programmable jouit actuellement d'un intérêt croissant dans l'industrie de la microélectronique. Comme nous l'avons mentionné dans le chapitre 1, les FPGA sont en train de rattraper les ASIC au niveau de la performance et de la densité d'intégration (et donc du prix), et sont plus avantageux que ces derniers en termes de temps et coûts de conception, ce qui explique leur succès actuel.

Il sera donc intéressant de surveiller si cette tendance se maintient dans les années à venir. En effet, si on est aujourd'hui capable de produire des ASIC en technologie 65 nm, on peut se demander combien de temps les FPGA, en dépit de leur différences, vont encore réussir à suivre les ASIC dans cette course vers ce qui deviendra sûrement un jour la nanoélectronique.

RÉFÉRENCES

ALTERA (2005). Altera. *www.altera.com.*

AMIRIX (2005). Ap100 pci platform fpga development board product datasheet. *Site Web Amirix.*

ARM (2005). Arm. *www.arm.com.*

BLYLER, J. (2004). Fpga prototyping tool aligns with asic flow. *Wireless System Design.*

BREBNER, G. (2002). Single-chip gigabit mixed-version ip router on virtex-ii pro. *IEEE Symposium on Field-Programmable Custom Computing Machines.*

CELOXICA (2003). Survey of system design trends. *Celoxica website.*

DENALI (2005). Denali. *www.denali.com.*

DUBOIS, M., BOIS, G. ET SAVARIA, Y. (2005). A double profiling methodology for a video processing platform.

ENDEAVOR (2005). Endeavor. *www.endeav.com.*

ESYS.NET (2005). Site web esys.net. *www.esys-net.org.*

EVE (2005). Eve. *www.eve-team.com.*

FISHER, R., PERKINS, S., WALKER, A. ET WOLFART, E. (2003). Feature detectors - sobel edge detector. *http ://home-pages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.*

FPGA-FAQ (2005). Fpga-faq. *www.fpga-faq.org.*

GHOZZI, F., NOUEL, P. ET MARCHEGAY, P. (2003). Acquisition et memorisation video sur systeme excalibur. *CCECE 2003, CCGEI 2003.*

HALL, T. ET HAMBLEN, J. (2004). System-on-a-programmable-chip development platforms in the classroom. *IEEE Transactions on Education*, 47.

IBM (2005). Ibm. *www.ibm.com.*

JERRAYA, A. ET NICOLESCU, G. (2003). *La spécification et la validation des systèmes hétérogènes embarqués.* Hermes.

KHALILOLLAHI, Y. (2004). What platform asics are and when to use them. *EE Times.*

KORDASIEWICZ, R. ET SHIRANI, S. (2004). Hardware implementation of the optimized transform and quantization blocks of h.264. *CCECE 2004, CCGEI 2004.*

KRISHNAN, G. ET THIRUMALAI, B. (2004). Fpgas drop below structured asic prices. *Xilinx Website.*

LI, J., PAPACHRISTOU, C. ET SHEKHAR, R. (2004). A reconfigurable soc architecture and caching scheme for 3d medical image processing. *IEEE Symposium on Field-Programmable Custom Computing Machines.*

LIPMAN, J. (2004). The missing link of soc design - platform and structured asics. *TechOnLine.*

MAHMUD, R. (2004). Prototyping an asic with fpgas. *www.fpgafaq.com.*

MENTOR (2005a). Mentor. *www.mentor.com.*

MENTOR (2005b). Mentor. *Seamless User's and Reference Manual.*

MICRIUM (2005). Micrium. *www.micrium.com.*

MORRIS, K. (2005). World's best fpga article - up to 53% more interesting. *www.fpgajournal.com.*

MUSEUM, I. V. (2005). Ieee virtual museum. *www.ieee-virtual-museum.org.*

OHBA, N. ET TAKANO, K. (2004). An soc design methodology using fpgas and embedded micropocessors. *DAC 2004.*

RABASTÉ, D. (2002). Asic et composants à réseaux logiques programmables : Pal, pld, cpld, fgpa.

RIZZATTI, L. (2004). New wave of fpga prototyping for system-on-chip designs. *www.techonline.com.*

SANCHO-PRADEL, D., JONES, S. ET GOODALL, M. (2002). System on programmable chip for real-time control implementations. *IEEE.*

SOUTHARD, J. (2004). Fpgas replacing asics in soc applications. *Mentor Website.*

SPACE (2005). Space. *www.grm.polymtl.ca/circus/fr/projets/space/index.html.*

STITT, G., LYSECKY, R. ET VAHID, F. (2003). Dynamic hardware/software partitioning : A first approach. *DAC.*

SUMMIT (2005). Summit. *www.summit-design.com.*

SYSTEMC (2005). Systemc. *www.systemc.org.*

TAKADA, H., HONDA, S., NISHIYAMA, R. ET YUYAMA, H. (2003). Hardware/software co-configuration for multiprocessor sopc. *IEEE Workshop on Software Technologies for Future Embedded Systems.*

U-BOOT (2005). Das u-boot - universal bootloader. *http ://sourceforge.net/projects/u-boot.*

WAKABAYASHI, K. ET OKAMOTO, T. (2004). C-based soc design flow and eda tools : An asic and system vendor perpective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits ans Systems*, 19.

XILINX (2005). Xilinx. *www.xilinx.com.*

ZAKHARIA, K. (2003). Your alternative to standard cell asic design has finally arrived. *The Syndicated.*