UNIVERSITÉ DE MONTRÉAL

ÉLABORATION D'UN MODÈLE D'ABSTRACTION DES COMMUNICATIONS POINT-À-POINT POUR UNE PLATEFORME (SOC) MULTIPROCESSEUR HÉTÉROGÈNE

SYLVAIN GOYETTE
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2008

Your file Votre référence
ISBN: 978-0-494-57252-8
Our file Notre référence
ISBN: 978-0-494-57252-8

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ÉLABORATION D'UN MODÈLE D'ABSTRACTION DES COMMUNICATIONS POINT-À-POINT POUR UNE PLATEFORME (SOC) MULTIPROCESSEUR HÉTÉROGÈNE

présenté par : GOYETTE Sylvain

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

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

M. LANGLOIS J.M. Pierre, Ph.D., président

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

M. ABOULHAMID El Mostapha, Ph.D., membre et codirecteur de recherche

M. BOLAND Jean-François, Ph.D., membre

« All truths are easy to understand once they are discovered; the point is to discover them. »

- Galileo Galilei

« A man's errors are his portals of discovery. »

- James Joyce

« New ideas pass through three periods: 1) It can't be done. 2) It probably can be done, but it's not worth doing. 3) I knew it was a good idea all along! »

- Arthur C. Clarke

REMERCIEMENTS

Je tiens à remercier M. Bois, mon directeur de maîtrise pour ses conseils, sa disponibilité et son analyse qui furent indispensables pour réaliser l'ensemble de mes travaux de recherche.

Je voudrais aussi souligner le support financier du CRNSG et du ReSMIQ qui me permit de me consacrer entièrement à ce projet. De plus, je remercie les membres du groupe de recherche en codesign CIRCUS de l'École Polytechnique, Cédric, Sébastien, Laurent, Luc, Maxime, Marc-André et Benoît, pour leur collaboration, leur soutien et les précieuses discussions.

Finalement, je tiens à remercier mes amis, mes parents et ma fiancée Trinh qui m'ont soutenu moralement tout au long de ces six longues années à Polytechnique.

RÉSUMÉ

Le domaine des systèmes embarqués fait face depuis quelques temps aux difficultés posées par leur complexité, l'écart de productivité entre la technologie et les outils de conception ainsi que les temps de commercialisation toujours plus courts. L'apparition des systèmes multiprocesseurs sur puce (MPSoC), où coexistent sur un même circuit différents composants logiciels et matériels, pose de nouveaux défis au niveau des communications inter-composants et de l'hétérogénéité des composants.

Un des aspects importants de la complexité inhérente des MPSoC est le choix de la topologie de l'architecture de communication. Plusieurs groupes de recherche dans le domaine travaillent au développement de langages, de plateformes et de modèles permettant d'augmenter le niveau d'abstraction des mécanismes de communication.

La symbiose compliquée du logiciel et du matériel impose une revue des méthodologies de conception. Le « Electronic System Level », ou ESL, propose un processus de conception et de vérification ciblant une abstraction à haut niveau qui permet d'améliorer la compréhensibilité et la facilité d'implémentation des systèmes. S'inscrivant dans cette pensée, la technologie de Space Codesign™, basée sur SystemC, assure une spécification unique et homogène des systèmes ainsi qu'un raffinement progressif vers des partitions logicielles et matérielles. Elle maintient la transparence des mécanismes de communication peu importe la technologie sous-jacente utilisée.

La recherche effectuée se base sur la technologie de Space Codesign™ et cible l'élaboration d'un modèle d'abstraction des communications point-à-point faisant bon usage des nouvelles avenues qu'offrent les processeurs embarqués (canaux haut-débit, extensibilité du jeu d'instruction, etc.). Ceci passe par la consolidation de la plateforme SPACE autour d'une architecture hétérogène basée sur le standard CoreConnect d'IBM

et des processeurs embarqués μBlaze et PowerPC 405 FX offert sur les FPGA Virtex-4 de la compagnie Xilinx.

Le paradigme résultant est le DirectLink qui augmente les fonctionnalités de la plateforme SPACE en étendant les services de communication existants basés sur la transmission de messages. DirectLink offre une interface unique tant pour les composants logiciels que matériels et diminue ainsi la complexité de la conception d'une topologie de système de communication.

DirectLink aide à améliorer les performances de l'architecture de communication et reste indépendant de la plateforme matérielle cible. Il peut diminuer les délais induits par les protocoles standards SPACE d'environ 90% et accélérer l'exécution d'une application complexe d'un facteur de 5. Le paradigme peut même s'étendre vers d'autres plateformes comme le Xtensa de Tensilica ou le NIOS-II d'Altera.

ABSTRACT

The field of embedded systems has been facing for some time now difficulties posed by an increasing gap of productivity between technology and design tools, an unprecedented complexity as well as increasingly short time to market. The apparition of multiprocessor systems-on-chip (MPSoC), in which software and hardware components coexist on the same circuit, poses new challenges in terms of inter-components communications and heterogeneity of components.

One of the important aspects of the inherent complexity of MPSoC is the choice of a topology for the communication architecture. Several research groups in the area are working on the development of languages, platforms and models to increase the level of abstraction of these communication mechanisms.

The intricate symbiosis between software and hardware requires a rethinking of design methodologies. ESL design and verification proposes a high-level abstraction that improves the understandability and ease of implementation of systems. In line with this thinking, the Space Codesign™ technology, based on SystemC, provides a unique and homogeneous system specification and a progressive refinement to software and hardware partitions. It maintains transparency of communication mechanisms regardless of the underlying technology.

This research is based on the Space Codesign™ technology and aims to develop of a model of abstraction for point-to-point communications making good use of new opportunities offered by embedded processors (high-speed channels, ISA extensibility, and so on.). This requires a consolidation of the SPACE platform around a heterogeneous architecture based on IBM CoreConnect standard and embedded

processors such as μBlaze and PowerPC 405 FX, all available on Virtex-4 Fogs manufactured by Xilinx.

The resulting paradigm is called DirectLink which increases the functionality of the platform by extending existing SPACE communication services based on message passing mechanisms. DirectLink offers a single interface for both software and hardware that reduces the complexity of communication system design.

DirectLink improves performances of the communication architecture and remains independent of the targeted hardware platform. It can reduce by approximately 90% delays induced by standard SPACE protocols and accelerate the execution of a complex application by a factor of 5. The paradigm may even extend to other platforms such as the Tensilica Xtensa or Altera NIOS-II.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
APU	Auxiliary Processor Unit
ASIC	Application Specific Integrated Circuit
BA	Bit Accurate
BCA	Bus Cycle Accurate
BRAM	Block RAM
CPLD	Complex Programmable Logic Device
DMA	Direct Memory Access
DCR	Device Control Register
EIC	External Interrupt Controller
ESL	Electronic System Level
FCB	Fabric Coprocessor Bus
FCM	Fabric Coprocessor Module
FIFO	First-In First-Out
FIT	Fixed Interval Timer
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Serial Link
GALS	Globally Asynchronous Locally Synchronous
GDB	GNU Debugger
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine

ISS	Instruction Set Simulator
MMU	Memory Management Unit
MPSoC	Multi-Processor System-on-Chip
OCM	On-Chip Memory
OEA	Operating Environment Architecture
OPB	On-chip Peripheral Bus
PIC	Programmable Interrupt Controller
PIT	Programmable Interval Timer
PLB	Processor Local Bus
PPC	PowerPC
RAM	Random Access Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
SDL	Space Direct Link
SISDL	Space In-Stream Direct Link
SOSDL	Space Out-Stream Direct Link
SPACE	SystemC Partitioning Aspects of Codesign and Exploration
SoC	System-on-chip
TLB	Translation Look-aside Buffer
TLM	Transaction-Level Modeling
TF	Timed Functional
UART	Universal Asynchronous Receiver/Transmitter
UDI	User-Defined Instruction
UEA	User Environment Architecture
UTF	Un-Timed Functional
VEA	Virtual Environment Architecture

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Cette section introduit le concept de systèmes sur puce (SoC), élément central de ce mémoire. Elle traite de maints aspects de leur conception, des méthodes de design et des architectures de communications utilisées.

1.1. Les systèmes embarqués modernes

Les systèmes embarqués sont de nos jours rendus omniprésents. L'environnement technologique dans lequel l'homme moderne baigne en est submergé. Leurs domaines applicatifs sont extrêmement variés, allant de l'appareil d'usage quotidien – téléphone cellulaire, montre digitale, four micro-onde – jusqu'aux gadgets technologiques et moyens de transports – iPod, système GPS, anti-patinage sur les voitures de luxe.

Considérant cela, il n'est pas surprenant d'apprendre que moins de 2% des 9 milliards de microprocesseurs et microcontrôleurs fabriqués en 2005 fassent partie des ordinateurs personnels et des stations de travail. Les systèmes embarqués ont utilisés les 8.8 autres milliards [1]. L'architecture d'un système embarqué peut varier du simple microcontrôleur exécutant son code d'une ROM jusqu'aux microprocesseurs roulant de complexes tâches en temps réel. Les variantes sont aussi nombreuses que sont les problèmes qu'elles ciblent.

Il ne serait pas faux d'affirmer que pratiquement tous les appareils électroniques actuellement produits sont des systèmes embarqués et que tous et chacun les utilisent dans son quotidien. En 2002, on estimait qu'une voiture contenait entre 40 et 50 microprocesseurs embarqués [2].

Comme la pression du marché continue de pousser le développement à des niveaux d'intégration de plus en plus élevés tout en tentant de réduire les coûts de productions, la taille des circuits et la consommation de puissance, l'innovation devient primordiale. Le Tableau 1.1 montre la croissance des marché et les prévisions jusqu'en 2009 [3]. Pour accommoder la croissance exponentielle de la demande pour des circuits de plus en plus gros, les concepteurs ont sans cesse poussé les limites des contraintes de la technologie et du design. Les systèmes embarqués sont le résultat de ce processus.

Tableau 1.1 – Marché des systèmes embarqués jusqu'en 2009 (Millions de $)

	2003	2004	2009	AAGR%[1]
Logiciels embarqués	1 401	1 641	3 448	16.0
Circuits intégrés embarqués	34 681	40 539	78 746	14.2
Cartes embarquées	3 401	3 693	5 950	10.0
Total	39 483	45 873	88 144	14.0

[1]Taux de croissance annuel moyen (Average Annual Growth Rate)

Comme mentionné précédemment, poussés par la demande, ces systèmes deviennent de plus en plus complexes. La technologie disponible permet l'accroissement de cette complexité car la quantité de transistors sur une puce suit une progression géométrique, doublant à tous les 18 mois selon la loi de Moore.

1.2. Problématique

La conception de systèmes embarqués devient une tâche de plus en plus ardue et les ingénieurs doivent résoudre d'imposants problèmes. Poussée par la forte pression du marché et la part toujours grandissante du contenu logiciel des architectures, la complexité des systèmes croît à chaque nouvelle génération. Ceci met à rude épreuve l'ingéniosité des concepteurs qui doivent souvent se battre avec des outils de design n'offrant pas des méthodologies à la productivité adéquate pour la tâche à réaliser.

Un problème important des systèmes embarqués est celui de l'exploration architecturale et plus particulièrement celui du partitionnement matériel-logiciel qui vise la

décomposition d'une spécification au niveau transactionnel. Le résultat du processus est une partition. Le but principal du concepteur d'un système est de trouver la partition qui requiert le moins de matériel pour rencontrer les critères de performance. Un aspect important de l'exploration architecturale et du partitionnement est le système de communication qui affecte grandement les performances d'un système donné.

Il est aujourd'hui devenu essentiel d'offrir aux concepteurs des outils qui accélèrent et facilitent la conception de tels systèmes embarqués, surtout dans le contexte très complexe des systèmes multiprocesseurs hétérogènes sur puces. Les environnements de développements actuels parviennent avec beaucoup de difficultés à offrir des solutions acceptables à ces obstacles.

La conception à bas niveau d'abstraction, tant sur le plan logiciel que matériel, demande trop de temps et d'efforts. Avoir à prendre en compte tous les détails d'implémentation d'une application C comportant plusieurs dizaines de milliers de lignes de code en même temps que la conception au niveau RTL d'une architecture matérielle pouvant contenir des millions de transistors devient un obstacle grandissant.

Les processus de conception doivent donc abstraire l'implémentation finale lors de la spécification et la conception du système. Les dépendances vers les détails de bas niveau doivent être éliminées ou réduites au maximum tout en laissant au concepteur un degré de liberté acceptable. De plus, la validation de cette spécification et de cette conception doit pouvoir commencer à être faite dès le début du processus en fournissant au concepteur des méthodologies et des abstractions qui fonctionneront correctement lors du raffinement à bas niveau.

De plus, du moment où la quantité de composants d'un MPSoC augmente, la largeur de bande de l'architecture de communication devient un facteur fondamental à considérer pour les performances globales. Il en résulte un besoin d'architectures de

communication à haut débit et extensibles. Cependant, la tendance de progression technologique indique que la performance et la consommation d'énergie des réseaux d'interconnexion se dégradent rapidement par rapport aux composants logiques [4]. Ceci résulte en un écart de performance significatif entre les réseaux de communication et la logique qui continuera à s'agrandir même en la présence de nouveaux modèles ou d'optimisations agressives desdits réseaux de communication..

L'environnement d'exploration architecturale et de conception SPACE [5], se basant sur la bibliothèque de classes SystemC, a pour but à résoudre ces nombreux problèmes. Il permet une abstraction complète des détails d'implémentation des systèmes-sur-puce lors des phases de design tout en réduisant les temps de conception et de validation. De plus, il facilite l'optimisation des performances.

1.3. Objectif

Le principal objectif de ce travail consiste à augmenter les fonctionnalités de la plateforme de conception SPACE en développant de nouveaux mécanismes de communications point-à-point propices aux architectures multiprocesseurs hétérogènes. Ces nouveaux mécanismes doivent tirer avantage de la présence de canaux de communication dédiés sur les processeurs embarqués ou la possibilité de configurer ces processeurs et se présenter sous la forme d'une abstraction complète des implémentations possibles.

Le paradigme choisi doit étendre les services de communication existants qui se basent sur la transmission de messages et aider à améliorer les performances de l'architecture de communication tout en restant générique et indépendant de la plateforme matérielle choisie. Le mécanisme doit être configurable afin d'offrir une certaine liberté de conception et offrir un compromis entre flexibilité et performances.

Afin de respecter la philosophie de SPACE, le lien point-à-point doit être offert pour des modules pré-partitionnés en matériel et en logiciel. Un lien direct point-à-point peut être aisément visualisé en matériel, mais ce n'est pas le cas pour un lien matériel/logiciel ou logiciel/logiciel multiprocesseur, pour lequel il faut prendre en compte les spécificités du processeur.

Toutes les prémisses de SPACE doivent être respectées, notamment concilier la souplesse du concept de glisser-déposer des modules dans SPACE aux caractéristiques d'une communication point-à-point.

1.4. Méthodologie

La réalisation de ces objectifs passe par maintes étapes :

1. Pour débuter, une familiarisation avec la librairie **SystemC** et l'environnement **SPACE** est nécessaire. Le concept du **TLM**, i.e. la modélisation au niveau système et transactionnelle, doit aussi être approfondi car c'est sur lui que repose la méthodologie de SPACE.
2. Par la suite sera définie une plateforme au niveau TLM qui peut recevoir un ou plusieurs processeurs configurables de types différents. L'environnement SPACE sera donc consolidé autour de l'architecture de communication CoreConnect d'IBM. La création et l'ajout d'un simulateur d'instruction du processeur configurable PowerPC405 FX sera l'étape logique subséquente afin d'avoir un environnement de conception complet couvrant l'ensemble des technologies disponibles par Xilinx. Cette plateforme permettra de prouver que le paradigme du lien point-à-point à haut niveau est valide et applicable sur différentes technologies.
3. Viendra ensuite la conception de l'interface de programmation à travers laquelle le nouveau mécanisme de communication sera disponible. Tel que requis par

SPACE, cette interface développée avec SystemC devra être utilisable à la fois par les tâches logicielles et matérielles.

4. L'étape subséquente consistera à développer des composants architecturaux nécessaires à l'implantation du nouveau mécanisme utilisant les différentes technologies disponibles sous la plateforme de simulation (actuellement les FPGA de Xilinx). Ces blocs matériels seront conçus au niveau transactionnel tout en gardant en tête que des versions synthétisables, décrites en VHDL, doivent être réalisables. La couche logicielle d'abstraction du matériel des différents processeurs sera modifiée pour tenir compte du nouveau mécanisme de communication point-à-point.
5. Une classe basée sur le patron de conception *Abstract Factory* sera ensuite développée afin de générer l'implantation du lien de communication point-à-point selon les blocs matériels ou logiciels qui y sont attachés. Cette classe abstraira le nouveau mécanisme peut importe la technologie utilisée et cachera la complexité de l'implémentation. Elle représentera un nouveau paradigme de communication dans SPACE peut importe le niveau d'abstraction du modèle.

1.5. Contributions

Ce travail a offert trois principales contributions :

1. D'abord, ce travail a permis une exploration de la littérature scientifique sur les méthodes de conception d'architectures de communication développées pour les systèmes multiprocesseurs sur puce, dans le contexte d'utilisation de processeurs configurables et extensibles.
2. Une nouvelle interface de programmation est désormais disponible dans l'environnement SPACE afin d'établir des communications point-à-point entre les différents modules logiciel et matériel dans une architecture multiprocesseur hétérogène. Un modèle d'abstraction de ces liens point-à-point est disponible

pour cacher les différentes implémentations possibles selon les technologies offertes dans SPACE. Ce modèle est générique et extensible à toute nouvelle technologie qui pourrait être introduite dans SPACE. Ces nouveaux mécanismes permettent d'améliorer l'utilisation de la bande passante disponible et d'accélérer les performances des applications.

3. De plus, la plateforme CoreConnect d'IBM est entièrement intégrée et consolidée dans l'environnement SPACE, augmentant ses capacités, son potentiel commercial et permettant de nouvelles possibilités lors de la conception de systèmes multiprocesseurs hétérogènes. Avec l'introduction dans SPACE d'un simulateur d'instruction du PowerPC405, la porte est ouverte pour des travaux futurs sur la modélisation et l'analyse de plateformes hétérogènes.

1.6. Organisation du mémoire

Ce mémoire est constitué de **cinq chapitres**. Le présent chapitre est une introduction aux systèmes embarqués ainsi qu'à la méthodologie et les objectifs de recherche. Le **deuxième** chapitre est une revue des différentes architectures de communication et des techniques utilisée pour déterminer l'architecture appropriée pour une plateforme multiprocesseur. Le **chapitre 3** constitue une description de l'intégration et de la consolidation d'une plateforme multiprocesseur hétérogène extensible dans un environnement de codesign logiciel/matériel. Le **chapitre 4** présente les objectifs et la conception du paradigme du DirectLink au niveau système dans la plateforme SPACE ainsi que les mécanismes utilisés dans les différentes technologies offertes dans cette plateforme. Le **dernier chapitre** présente les performances du DirectLink dans différentes technologies et des exemples de gains dans des applications. Enfin, la conclusion résume le travail de recherche qui a été accompli et suggère de futurs travaux. Afin de maintenir la concision de ce mémoire et puisque beaucoup de projets de recherche du laboratoire CIRCUS ont déjà abordé maintes fois certains thèmes et sujets, plusieurs sections sont présentées en annexe.

CHAPITRE 2

LES COMMUNICATIONS DANS UN SYSTÈME-SUR-PUCE

Avec l'apparition d'année en année de circuits intégrés offrant des centaines de millions de transistors, les systèmes multiprocesseurs sur puce gagnent en intérêt. Le concept de système-sur-puce (SoC), apparu au tournant du millénaire, se veut la réunion complexe sur une seule puce de différents composants hétérogènes dans le but de répondre aux besoins croissants de performance du marché dans des domaines d'application spécialisés comme le multimédia, des télécommunications, de l'automatisation, etc.

Ce chapitre présente dans un premier temps les différents modèles et architectures de communication pour les MPSoC. Par la suite, un survol de quelques paradigmes d'abstraction des communications à haut niveau et des méthodologies de génération d'interfaces logicielles/matérielles est présenté.

2.1. Les modèles de communication pour MPSoC

Les communications dans un MPSoC se classifient en 3 catégories distinctes selon les partitions impliquées.

2.1.1. *Communication inter logiciel*

Dans les MPSoC, les communications logiciel/logiciel peuvent se produire entre deux tâches qui se retrouvent sur un même processeur ou sur deux processeurs différents. De plus, les communications peuvent être implicites ou explicites [32]:

- **Communication implicite** : s'effectue par un mécanisme de mémoire partagée et visible par plusieurs tâches ou plusieurs microprocesseurs. Des mécanismes de

synchronisation sont donc nécessaire afin d'assurer la cohérence et l'intégrité des données partagées.

- **Communication explicite** : s'effectue par des appels à des méthodes du type écriture ou lecture, bloquantes ou non bloquantes. Ce mécanisme se base sur le concept de passation de messages et implique donc l'utilisation d'un FIFO logiciel ou matériel.

2.1.2. Communication inter matériel

L'architecture de communication pour la partition matérielle a un impact significatif sur les performances, la taille du circuit, la dissipation de puissance et le coût global. De plus, les mêmes mécanismes de communication explicite et implicite mentionnés en 2.1.1 peuvent être utilisés pour les échanges inter matériel.

2.1.3. Communication logiciel/matériel

Dans les MPSoC, ce type de communication représente les échanges de données entre une tâche logicielle s'exécutant sur un processeur et un module du reste de la plateforme matérielle. Les données peuvent transiger à travers les mêmes mécanismes que pour les communications inter matérielles. Cependant, la liaison entre la tâche logicielle et l'interface matérielle n'est pas directe car les données seront souvent traitées par des fonctions de la pile logicielle qui abstraient le protocole de communication matériel utilisé.

De plus, il peut y avoir des mécanismes de communication directs, propres au microprocesseur utilisé, qui permettent d'établir un lien dédié entre le processeur et un module matériel. Le processeur peut aussi offrir une interface pour connecter un coprocesseur directement au cœur du processeur. Dans le cas des données qui transigent du matériel vers le logiciel, i.e. une tâche s'exécutant sur un processeur, deux

mécanismes peuvent être utilisés : les interruptions ou l'attente active (de l'anglais *polling*).

2.2. Les architectures de communication

Selon [33] la taxonomie des réseaux d'interconnexion de SoC s'opère selon une classification en arbre – tel qu'illustré à la Figure 2.1– basée sur leur structure physique, les protocoles de communications et les interfaces utilisées. L'emphase sera ici mise sur les bus, les connections point à point et le concept de GALS. Le domaine des NoC (ou *Network on Chip*, réseau sur puce) ne sera pas exploré ici puisqu'il n'intervient pas dans la compréhension des thèmes qui seront abordés ultérieurement.

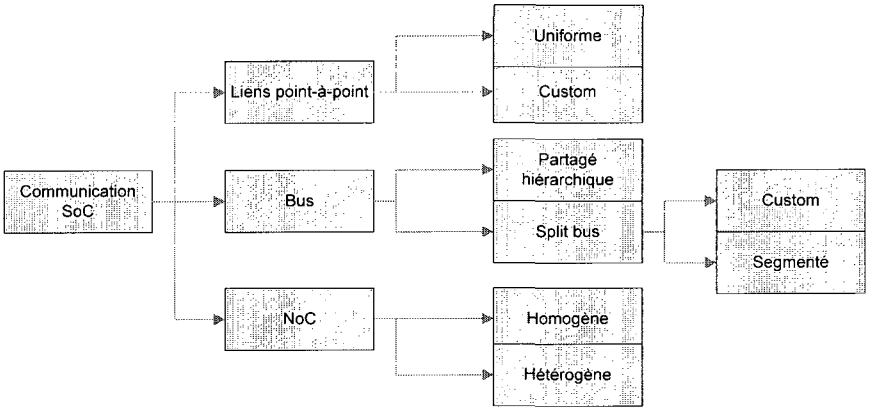


Figure 2.1 – Taxonomie des architectures communication pour SoC

2.2.1. Bus de communication

Dans un bus, les connections physiques (fils) sont groupés pour former un seul canal de communication physique qui est partagé entre différents canaux logiques. Cette architecture diminue la longueur totale de fils nécessaire et diminue la surface requise pour les interfaces. Cependant, avec la réduction d'échelle continue des circuits, plus il y a de blocs attachés au bus, plus il est difficile de gérer la longueur des fils lors du

placement et routage [33]. Les bus permettent la communication entre différents blocs matériels d'une architecture qui sont séparés en deux catégories : maîtres (qui initient les transactions) et les esclaves (qui exécutent les transactions).

La bande passante d'un bus peut être améliorée en utilisant un bus matriciel (ou *cross-bar switch*) où chaque composant maître est connecté à tous les composants esclaves et dans lequel plusieurs canaux parallèles sont disponibles pour supporter des transmissions concurrentes de données, e.g. le bus Avalon d'Altera [9] ou AMBA de ARM [34]. L'architecture d'un bus se base sur 3 paramètres [32] visant à répondre aux objectifs systémiques :

- **Arbitrage :** Puisque plusieurs maîtres peuvent initier des transactions sur un bus, un mécanisme d'arbitrage utilisant une méthode appropriée est nécessaire. Le mécanisme affecte le taux d'utilisation et la latence. Les méthodes *First-In-First-Out, Round-Robin* – chances égales d'obtenir le bus – ou basés sur les priorités sont souvent utilisées.
- **Fréquence et largeur du bus :** Déterminent le débit des transferts et influencent la consommation d'énergie, la surface requise et le coût.
- **Types de transferts :** Un bus peut implanter des mécanismes de transfert plus complexes que les simples lectures et écritures de mots de donnée :
    - Blocs fixes : transfert de bloc dont la taille est une puissance de 2, souvent utilisé pour les mémoires caches.
    - Différés (*split transaction*) : un transfert peut être interrompu par une requête d'un autre maître puis reprendre lorsqu'elle est complétée.
    - Atomique : un mécanisme de réservation du bus (*bus lock)* assure à un maître l'obtention ininterrompue du bus pour des transferts en rafale.

2.2.2. Liens point à point et communication ad hoc

Dans un lien point à point, ou direct, une paire de blocs de calcul communiquent directement à travers des connections (fils) physiquement dédiées. Cette simplicité est l'avantage majeur de cette architecture. Il en existe certaines topologies uniformes comme les réseaux de neurones [33]

Il permet de diminuer la latence de communication entre deux modules matériels et réduire la puissance utilisée. La latence et la performance deviennent déterministes. Le transfert de données se fait directement d'un module à un autre sans passer par un canal partagé. On élimine donc la complexité et la latence induite par l'arbitrage et les signaux de contrôle d'un bus. Un désavantage flagrant est qu'il limite la réutilisation de l'architecture en impliquant un design spécifique à l'application. De plus, il devient difficile de concevoir la topologie d'un système où les modules communiquent avec plusieurs autres composants si uniquement des liens directs sont utilisés. Le taux d'utilisation des connections physiques est bas dans le cas d'application à faible bande passante et une interface dédiée est requise pour chaque canal de communication.

Les données transmises peuvent être tamponnées ou non. Dans le cas d'utilisation d'un tampon (FIFO), le mécanisme permet la lecture/écriture bloquante et non bloquante selon l'état du FIFO (vide ou plein). Il est donc possible d'établir un système de communication de passation de message avec rendez-vous. Dans le cas de communication bidirectionnelle, il est nécessaire d'avoir 2 liens directs.

2.2.3. GALS

Le concept des GALS (*Globally Asynchronous Locally Synchronous*) a été proposé par Chapiro en 1984 [35]. Ce paradigme permet la réduction de l'effort de conception du réseau de distribution d'horloge, une diminution de la consommation de puissance et la

réutilisation de blocs IP conçus et optimisés pour fonctionner à une fréquence spécifique [36]. La Figure 2.2 illustre l'organisation globale d'un système GALS qui relie différents domaines d'horloge via un mécanisme d'interconnexion asynchrone [37] et qui sera selon [38] l'implémentation de la prochaine génération de SoC pour atteindre les requis de contrainte de puissance.

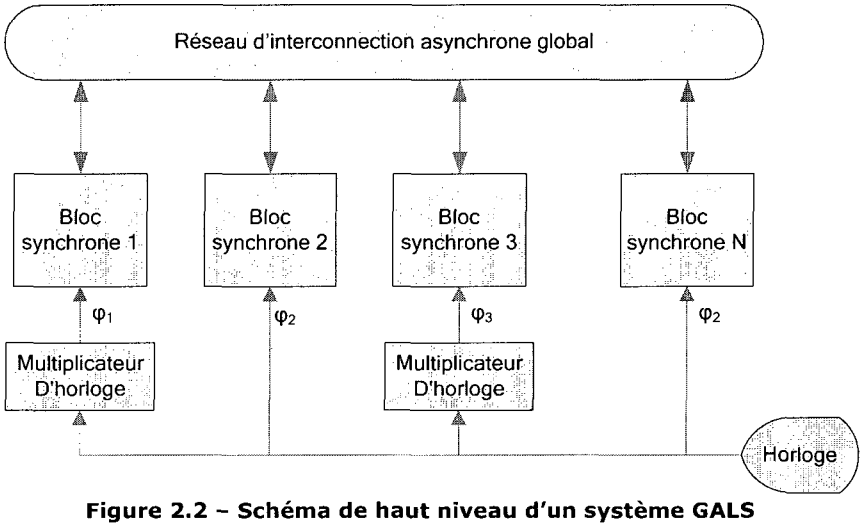


Figure 2.2 – Schéma de haut niveau d'un système GALS

Une méthodologie de partitionnement a été développée dans [39] pour optimiser la consommation de puissance d'une architecture en se basant sur les quatre modes de communication possible des GALS : FIFO, *handshake*, *strobe*, ou *send and forget*.

Dans [40], les auteurs proposent une interface GALS à basse latence et ne requérant pas d'altérer les signaux d'horloges, qui se base sur des FIFOs asynchrones dans un contexte multiprocesseur et de communication avec un coprocesseur. Chaque sous-système uniprocesseur est connecté aux autres par des ponts asynchrones tels qu'illustré à la Figure 2.3.

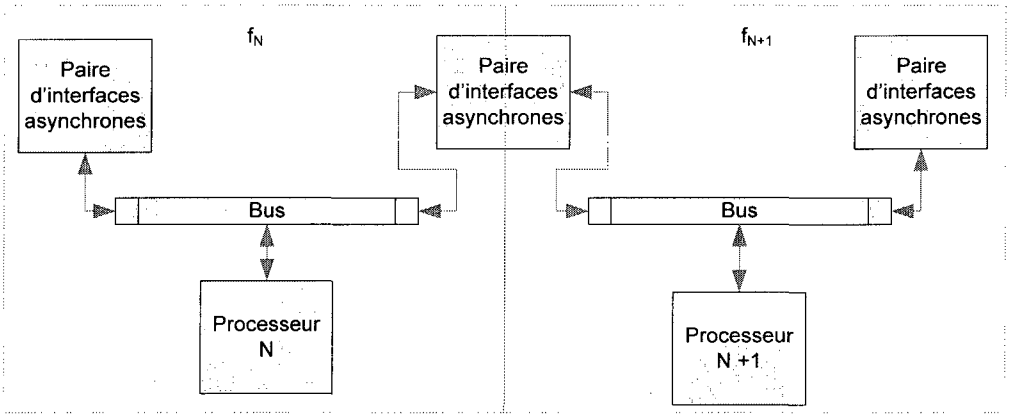


Figure 2.3 – Interconnexion asynchrone de sous-systèmes uniprocesseur

La méthodologie de [41] adresse le problème de l'analyse de performance et de puissance des systèmes GALS en utilisant une modélisation au niveau transactionnel basée sur SystemC. Les processus de calcul y sont décrits dans un langage procédural (comme le C) et interfacent un canal de communication BCA modélisant une FIFO asynchrone.

2.3. Abstraction des communications à haut niveau

L'abstraction des architectures de communication à haut niveau est un des avantages de la méthodologie TLM. Quelques modèles existant sont ici décrits.

2.3.1. Le modèle SPACE

La plateforme SPACE (ANNEXE D) offre un modèle d'abstraction qui permet d'assurer la communication entre les modules logiciels et matériels d'une architecture par un mécanisme de passation de messages.

2.3.1.1.Paradigme de communication

SPACE utilise un paradigme de communication asynchrone par passage de message. Les messages envoyés sont stockés dans des FIFO jusqu'à la leur lecture. Chaque message comporte deux parties : un en-tête de quatre octets contenant des drapeaux, la taille du message et les identificateurs numériques des modules source et destination, ainsi que la charge utile du message, soit les données. La taille du message peut être de 4 à 256 octets. Les envois sont sérialisés par les adaptateurs afin de pouvoir transiger sur le bus partagé.

Ces mécanismes sont assurés par la présence de primitives de communications accessible par les processus légers des modules SPACE aussi bien logiciels que matériels. La complexité de l'implémentation est donc abstraite et indépendante de la fonctionnalité d'un module. Il faut toujours respecter le modèle de communication par rendez-vous, e.g. si un module *A* écrit un message à *B,* il doit utiliser une primitive de lecture et vice versa.

2.3.1.2.Les primitives de communication

SPACE fournit quatre fonctions de base pour communiquer avec un autre module ou un périphérique de SpaceLib. L'implémentation utilise le bus partagé, les adaptateurs matériels et une couche spéciale de la pile logicielle. Ces fonctions sont les suivantes :

- **ModuleWrite :** Permet l'écriture d'un message d'un module vers un module. Les messages transigent par le bus et sont stockés dans des FIFO matérielles ou logicielles jusqu'à ce que le destinataire le consomme. Les messages provenant du matériel vers le logiciel sont stockés dans l'ISSAdapter qui notifie le processeur par interruption de leurs arrivées.

- **ModuleRead :** Permet la lecture d'un message qui a été stockée dans une FIFO logicielle (ex. : liste chaînée en mémoire) ou matérielle (ex. : lien FSL). C'est une opération locale entre un module et son adaptateur ne nécessitant pas d'opération sur le bus partagé.
- **DeviceWrite :** Écriture à un périphérique par le bus.
- **DeviceRead :** Lecture d'un périphérique par le bus.

Les champs OrgID et DestID sont inclus dans les en-têtes de message (Tableau 2.1) et permettent d'effectuer le routage des messages dans l'architecture de communication. L'origine sert à stocker le message dans la FIFO appropriée quand un module de destination peut recevoir des messages de plusieurs sources. Aussi, le champ Timeout est encodé dans la portion *flags* de l'en-tête du message.

Tableau 2.1 – Paramètres des primitives de communication

Paramètre	Signification
OrgID	Identificateur numérique du module source
DestID	Identificateur numérique du module de destination
Priority	Priorité du message
*pData	Pointeur sur le tampon de mémoire contenant le message
DataSize	Taille du message
TimeOut	Paramètre indiquant si l'opération est bloquante ou non bloquante

2.3.1.3.Les types de communication

SPACE support quatre types de communication en fonction de la valeur du paramètre *TimeOut* qui peut être SPACE_WAIT_FOREVER (bloquant) ou SPACE_NO_WAIT (non bloquant). Les opérations avec expiration dans un délai de temps spécifié ne sont pas supportées pour le moment.

- **Lecture non bloquante :** Le module effectuant la lecture n'attend pas que des données soit présentes dans la FIFO avant de poursuivre son exécution.
- **Lecture bloquante :** Le module effectuant la lecture attend la présence d'un message dans la FIFO avant de poursuivre son exécution.
- **Écriture non bloquante :** Le module effectuant l'écriture poursuit son exécution immédiatement après avoir complété la transmission de son message.
- **Écriture bloquante :** Le module effectuant l'écriture attend que le message soit lu par la destination avant de poursuivre son exécution. Le module attend donc la réception d'un acquittement (*ACK*) en provenance du module de destination.

De plus, après la complétion d'une lecture (bloquante ou non bloquante), le module doit absolument envoyé un ACK à la source si le message a été envoyé par une écriture bloquante. Si une opération échoue, un code d'erreur sera retourné par la primitive de communication.

2.3.1.4.Les classes de communication

Selon les partitions impliquées dans un transfert de données au niveau Simtek, il existe quatre classes de communication se rapprochant des trois modèles de la section 2.1. Au niveau Elix, toutes les communications sont de la classe matériel/matériel.

- **Classe matériel/matériel :** Le message transige d'un module de la partition matériel vers un autre module de la même partition. Le transfert se produit en deux étapes :
    - Le module M1 appelle la primitive ModuleWrite. Le message est passé à l'adaptateur de communication A1 qui sérialise le message afin de le transférer sur le bus. L'adaptateur A2 réceptionne le message et le stocke dans une FIFO interne.

- Le module M2 appelle la primitive ModuleRead. Le message est lu directement dans la FIFO de l'adaptateur A2 via son l'interface. Il est à noter que M2 peut renvoyer un ACK à M1 si le ModuleWrite était bloquant. Le mécanisme de transmission est le même.

- **Classe logiciel/logiciel :** Le message transige d'un module logiciel vers un autre module de la même partition. Il y a deux cas de figure possibles :
  - Cas uniprocesseur : Les modules sont sur le même processeur, le mécanisme est donc interne au logiciel applicatif de ce processeur et fait intervenir un mécanisme de communication explicite interprocessus habituel (voir section 2.1.1) utilisant une FIFO logicielle. Les mécanismes du RTOS sont utilisés par les primitives de communication.
  - Cas multiprocesseur : Les modules sont sur des processeurs distincts. Ce cas revient à la classe logiciel/matériel. Le matériel est ici l'ISSAdapter associé au processeur contenant le module de destination.

- **Classe logiciel/matériel :** Un module logiciel écrit un message vers un module matériel. Le message transige directement à travers le bus via l'interface du processeur contenant le module logiciel M1 et est stocké dans la FIFO de l'adaptateur A2 du module matériel M2. Le module M2 lit localement le message dans la FIFO de son adaptateur. La primitive de communication ModuleWrite est définie dans une couche d'abstraction des communications de la pile logicielle.

- **Classe matériel/logiciel :** Un module matériel écrit un message à un module logiciel. Le processus se déroule en quatre étapes :
  - Le module M1 envoi le message à l'ISSAdapter du processeur (ISS) contenant le module M2 via son adaptateur et le bus partagé. L'ISSAdapter stocke le message dans une FIFO interne.

- L'ISSAdapter notifie par interruption (section 2.1.3) le processeur contenant M2.
- Une routine de service d'interruption (ISR) s'exécute et récupère le message pour le stocker dans une FIFO logicielle propre aux mécanismes de communication interprocessus (section 2.1.1).
- Le module M2 récupère le message dans sa FIFO logicielle.

2.3.1.5.Fonctionnement de la pile logicielle

Sous SPACE, le logiciel applicatif est composé des modules de la partition logicielle originalement décrits en SystemC avec les ajouts de SPACE pour les primitives de communication.

Ces modules sont interprétés comme étant des tâches logicielles s'exécutant sur un RTOS, ici µC/OS-II. Une API (*Application Programming Interface*) spécifiquement développée pour ce RTOS permet la conversion des appels de fonction SystemC en appels compris par le système d'exploitation (Tableau 2.2). Cet API se nomme **Tor** dans SPACE. Ceci permet de supporter, sans modifications aux modules logiciels, l'ordonnancement préemptif et la gestion multitâche sur un processeur exécutant plusieurs modules SPACE. Le Tor comporte aussi l'implémentation générique des mécanismes de communication SPACE.

Tableau 2.2 – Exemples de conversions d'appels SystemC en appels µC/OS-II

Fonction SystemC	Fonction MicroC/OS-II	Description
wait()	OSTaskDelay()	Attente d'un délai d'un nombre de cycles spécifié
sc_start()	OSStartScheduler()	Lancement de la simulation ou de l'ordonnanceur du RTOS
sc_stop()	OSStopScheduler()	Arrêt de la simulation ou de l'ordonnanceur du RTOS

2.3.1.6.Raffinement BCA actuel dans SPACE

Les adaptateurs de communications des modules ont à la fois une interface maîtresse et esclave sur le bus. Ainsi, ils peuvent initier des transactions et répondre aux requêtes des autres modules. Tous les adaptateurs implémentent la même interface unique (SpaceModuleIF) qui permet de les connecter sur le port de communication du module utilisateur. Puisque le bus utilise des adresses de 32 bits pour identifier les esclaves connectés, les adaptateurs de communication doivent effectuer une conversion des identificateurs numériques des modules. Une fonction permet de dynamiquement générer les adresses en décalant de 22 bits l'identificateur numérique stocké sur un octet (Tableau 2.3).

Tableau 2.3 – Structure d'une adresse dans SPACE

Bits 30-31	Bits 22-29	Bits 0-14
Bits prévus pour la gestion multibus	Identificateur numérique du module	Espace mémoire réservé pour le module (4 mégaoctets)

Cette méthode de conversion des identificateurs en adresses permet 3 choses :

- Abstraire le mode d'adressage des modules ;
- Réduire la taille des en-têtes des messages ;
- Rendre homogène l'identification des modules matériels et logiciels.

L'architecture de communication doit aussi assurer l'atomicité des transferts des messages afin d'éviter qu'un transfert vers un module soit interrompu par un autre transfert vers le même module. Ceci violerait l'intégrité du premier message en cours de transmission. Pour ce faire, les modules matériels activent physiquement le *bus lock* (voir section 2.2.1) pendant une communication. Les modules logiciels doivent quant à eux utiliser une instruction spéciale du processeur afin d'indiquer aux contrôleurs de bus d'activer le *bus lock.*

2.3.2. Le modèle ImpulseC

ImpulseC [15] étend, à la manière de SystemC, le langage ANSI C avec des librairies supportant la création d'applications basées sur les processus de communication séquentiels [42] (ou *Communication Sequential Processes, CSP*), i.e. un paradigme de programmation similaire au modèle flot de données. ImpulseC vise le développement d'**applications parallèles** qui peuvent aisément être converties en VHDL. Il permet que la description d'une application logicielle puisse être appliquée directement sur une plateforme matériel/logiciel.

Le paradigme de programmation est basé sur le transfert tamponné de données entre des processus via des flux (ou flots de données) gérés comme des FIFOs. Ceci permet l'écriture d'applications parallèles à haut niveau d'abstraction sans besoin de mécanismes de synchronisation entre les processus. Les différentes portions d'une application sont exécutées de manière concurrente et indépendamment synchronisés par des processus. Ils opèrent en acceptant des données en entrée, en opérant des calculs sur celles-ci et en générant des résultats en sortie. Ils sont persistants, invoqués qu'une seule fois et s'exécutant tant et aussi longtemps qu'il y a des données à traiter. Les données transitent de processus en processus le long des flux.

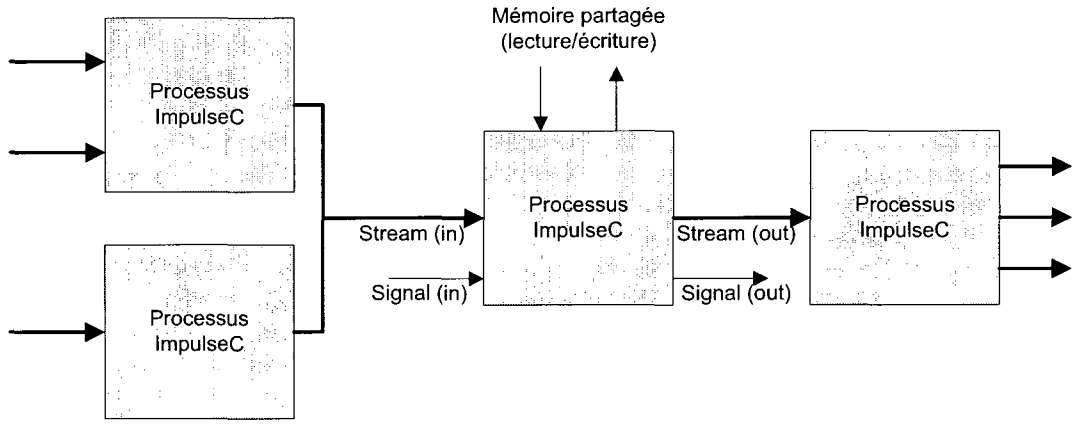


Figure 2.4 – Modèle d'application ImpulseC utilisant le paradigme CSP

L'API d'ImpulseC fournit des méthodes de communication incluant les *streams*, la mémoire partagée et les signaux (Figure 2.4). Les *streams* représentent des canaux de communication unidirectionnelle reliant des processus matériels ou logiciels. Ils sont définis par la largeur des mots de données qu'ils acceptent (de 8 bits à 128 bits) et la profondeur du tampon. On peut y effectuer des opérations de lecture/écriture via les fonctions **co_stream_read** et **co_stream_write**. Chaque flot de données est manipulé explicitement par les fonctions. La lecture bloque tant qu'il n'y a pas de données dans le canal et l'écriture bloque tant que le canal est plein. En matériel, les *streams* sont implémentés comme des FIFO qui peuvent avoir une ou deux horloges si les processus s'exécutent à des fréquences différentes. Le *stream* abstrait les détails des communications logicielles/matérielles pour les différents types de plateformes et rend l'application portable.

CoDeveloper est un environnement de développement supportant le modèle ImpulseC. Il permet le partitionnement logiciel/matériel des processus ImpulseC. CoDeveloper supporte une large variété de plateformes matérielles (Xilinx, Altera, etc.). Il génère automatiquement les modules VHDL, les interfaces matérielles et l'HAL en se basant sur une librairie de composants. Sur le Virtex-4 de Xilinx, les *streams* connectés aux processus s'exécutant sur le PowerPC405 Fx utilisent l'interface APU ou le PLB [43][44].

2.3.3. Le modèle d'interface matériel/logiciel unifié

Dans [11], les auteurs proposent une interface unifiée qui abstrait, du côté logiciel, le CPU sous une couche implémentant des pilotes de bases, des fonctionnalités I/O et des systèmes d'exploitation. Du côté matériel, cette interface abstrait les détails du bus du CPU. À la différence de SPACE, cette approche n'utilise pas deux modèles compilés séparément de l'interface, soit un pour le logiciel et un pour le matériel.

L'API (Figure 2.5) cache les détails matériels comme les contrôleurs d'interruption, la mémoire et les systèmes I/O. Le logiciel *hardware-dependant* support l'API et l'adapte au sous-système CPU. Ce sous-système élimine l'architecture complexe généralement présente dans un simulateur de jeu d'instruction (ou ISS, voir ANNEXE C) pour atteindre les performances de simulation requises. L'objectif est d'être capable de concevoir le matériel **et** le logiciel à tous les niveaux d'abstraction.

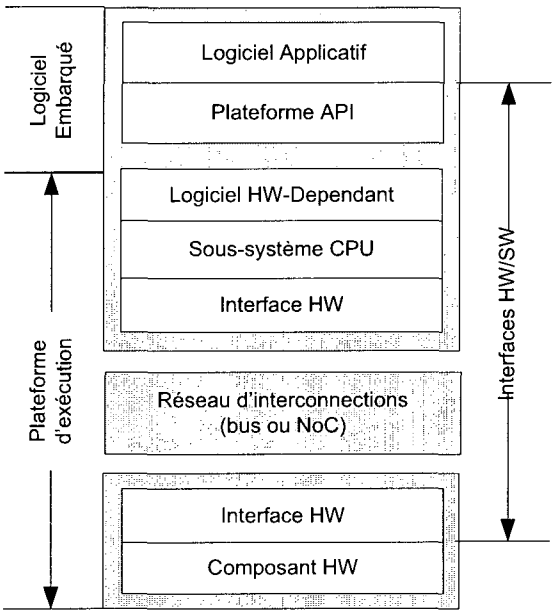


Figure 2.5 – Interface logicielle/matérielle unifiée

À haut niveau, un système est représenté par un ensemble de processus qui peuvent communiquer à travers une multitude de canaux abstraits. Un processus désirant communiquer à travers un canal précis doit appeler la primitive de communication (*send* ou *receive*) de ce canal. Chaque application peut posséder son propre ensemble de primitives (*send_short*, *send_atm*, *send_int*, etc.). Chaque canal se base sur des unités de communication sélectionnées d'une librairie. D'un point de vue conceptuel, une unité de communication est un objet qui peut exécuter une ou plusieurs primitives de communication avec un protocole spécifique [45] et à différents niveaux d'abstractions [46].

2.3.3.1.Modèle de simulation

Comparativement à une approche traditionnelle basée ISS, le sous-système CPU du modèle de simulation, utilisant SystemC, fournit un environnement où les composants logiciels peuvent s'exécuter nativement sur la machine hôte de simulation [47]. Le sous-système CPU implémente à la fois l'interface matérielle sous la forme de services fournis (ports SystemC slave) et de services requis (ports SystemC master), ainsi que l'API logiciel. Au lieu de définir en détail toute la mécanique interne d'un processeur (comme dans un ISS), le sous-système CPU implémente les concepts abstraits suivants dans un méta-modèle de processeur :

- **Unité d'exécution** : Fil d'exécution logiciel (thread) et les méthodes pour manipuler son exécution (changements de contexte) ;
- **Unité de données** : Emplacement pour le stockage de données et abstraction d'une entité physique comme la mémoire. Permet la lecture/écriture par adresse via des méthodes ;
- **Unité d'accès** : Abstraction de la projection en mémoire (*memory map*) d'une unité d'exécution. Pourrait implémenter des mécanismes de MMU tel la mémoire virtuelle ;
- **Unité de synchronisation** : Permet l'abstraction des interruptions et l'attachement d'ISR.

Le logiciel devant s'exécuter sur le processeur n'est pas interprété mais exécuté nativement. Puisque le logiciel est indépendant du matériel, il peut être compilé pour la machine hôte et manipuler la librairie de HAL (interface unifiée) correspondant au processeur à modéliser [48]. L'**inconvénient** est que le code logiciel doit être modifié par l'ajout d'annotations temporelles permettant de conserver la précision du temps de simulation tout en effectuant une simulation comportementale.

2.4. Génération des interfaces logiciel-matériel

Il existe différentes manières d'établir les interfaces de communication entre les partitions logicielles et matérielles.

2.4.1. L'approche des composants basés service

Dans [49] et [50], les auteurs proposent un modèle unifié qui généralise l'approche de composants basés service pour englober l'interface logiciel/matériel. Cette méthode permet une fine composition de l'implémentation de l'interface, un partitionnement flexible et une génération automatique de l'architecture basée sur une librairie de composantes. L'abstraction de cette interface est définie en spécifiant deux ensembles de services : (1) les services reliés au logiciel applicatif (API) et (2) les services reliés au réseau d'interconnexion (services de communication)

Chaque composant est modélisé par la déclaration d'une interface qui définit les services offerts et la manière de le connecter aux autres composants, et un arbre d'implémentation qui décrit le comportement du composant en utilisant du macrocode qui sera utilisé pour générer le logiciel, le modèle matériel ou le modèle de simulation fonctionnel. L'utilisation de ce modèle vise un flot de conception en 2 étapes:

- Conception et partitionnement de l'architecture à haut-niveau résultant en une abstraction du système comportant des modules logiciels, matériels et des interfaces abstraites de communication;
- Implémentation de chaque élément décrit dans le modèle abstrait.

L'outil de génération prend la spécification de l'interface et le modèle de composant en entrée afin de sélectionner un ensemble de composants nécessaire à l'implantation et

d'en créer des instances en les paramétrant selon la valeur des paramètres fournis à l'interface abstraite. La Figure 2.6 illustre cette méthode.

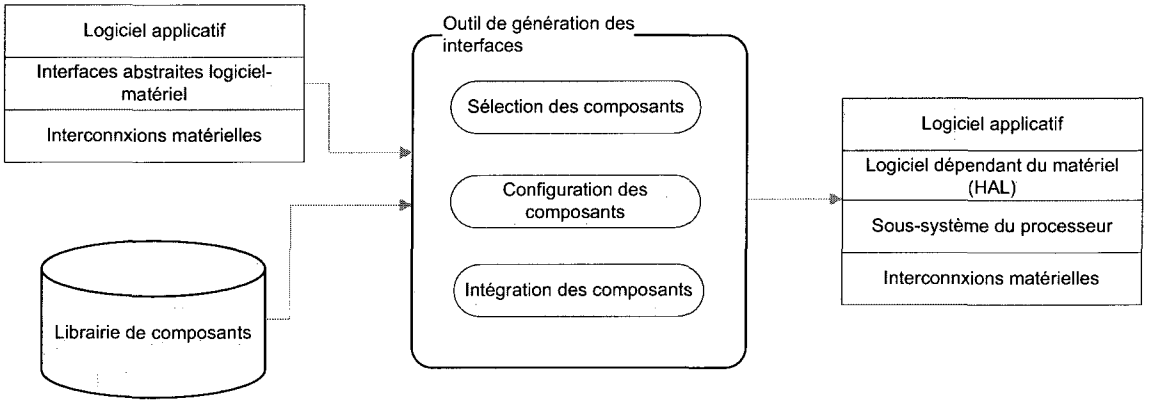


Figure 2.6 – Outil de génération des interfaces logicielles/matérielles

2.4.2. L'approche des adaptateurs de communication

Cette approche utilise des adaptateurs pour convertir les protocoles de communication à bas niveau entre les modules matériels et l'architecture cible. Une interface matérielle est aussi utilisée pour gérer les communications entre la partition logicielle (processeur embarqué) et le reste de l'architecture matérielle. Cet adaptateur de processeur peut se connecter directement au bus partagé de la plateforme ou à des canaux dédiés rapides du processeur. C'est l'approche utilisée dans SPACE.

2.4.3. L'approche coprocesseur

Dans cette approche, les modules de la partition matérielle sont implémentés comme coprocesseurs directement connectés au processeur exécutant la partie logicielle de l'architecture. Dans [51], la méthodologie proposée exige un profilage d'une spécification à haut niveau en C et convertit les fonctions qui demandent beaucoup de temps de processeurs en coprocesseurs décrit en VHDL. Des adaptateurs de coprocesseurs sont automatiquement insérés (basé sur une librairie) dans l'architecture

et permettent d'adapter chaque type de processeur à une interface unique de coprocesseur. Ce support utilise la possibilité d'étendre l'ISA (jeu d'instruction ou *Instruction Set Architecture*) du processeur ou des instructions de communication pour coprocesseur. L'adaptateur tamponne les données avant de les transmettre au coprocesseur.

2.4.4. L'approche du Tensilica Xtensa

Dans [28][52], Tensilica propose une méthode de connexion directe inter processeur pour réduire les coûts de communication et la latence.

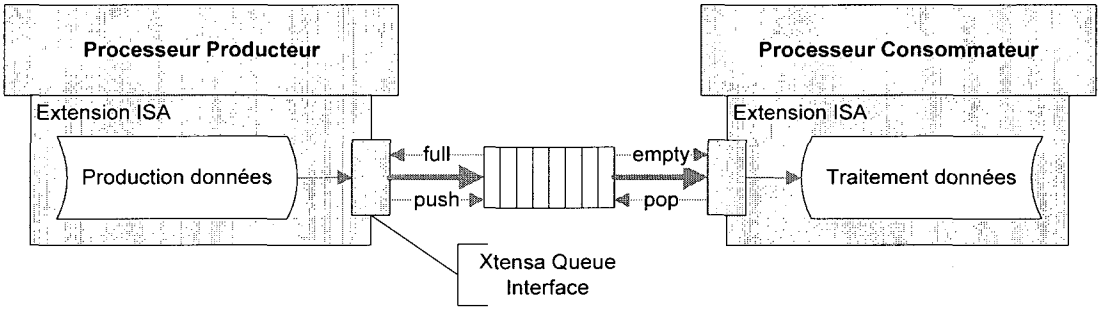


Figure 2.7 – Extension du Xtensa pour communication par queue de données

La Figure 2.7 illustre un exemple de connexion directe. Le mécanisme de communication tâche-à-tâche ayant la plus grande bande passante est l'implémentation matérielle de queues de données. Une FIFO peut soutenir un débit d'une donnée par cycle. Le *handshake* entre le consommateur et le producteur est implicite dans l'interface processeur/FIFO. L'opération d'écriture (*push*) bloque le producteur si la queue est pleine. L'opération de lecteur (pop) bloque le consommateur si aucune donnée n'est disponible. Les opérations non bloquantes sont disponibles si le développeur vérifie explicitement l'état de la queue avant d'effectuer un *push/pop*.

Les extensions de l'ISA du Xtensa permettent l'implémentation directe des FIFO. Une instruction peut spécifier une FIFO comme étant une des destinations pour ses résultats ou l'utiliser comme source de données, au lieu d'un registre. Une instruction peut performer plusieurs opérations sur les FIFO par cycles, en combinant par exemple deux FIFO en entrée et deux FIFO en sortie. Ceci permet une agrégation de la bande passante de plusieurs liens et une minimisation de la surcharge de contrôle. Un seul Xtensa LX peut posséder 300 interfaces de FIFO dont la largeur peut aller jusqu'à 1024 bits. En se basant sur ce concept, les auteurs de [53] proposent une conception d'architecture style pipeline pour des configurations de MPSoC hétérogènes en utilisant le Xtensa LX. Chaque configuration comporte un ensemble de cœurs Xtensa qui représente chacun une étape du pipeline. Les cœurs transfèrent leurs données via des FIFO, ce qui leur permet de s'exécuter indépendamment les uns des autres.

CHAPITRE 3

UNE PLATEFORME VIRTUELLE HÉTÉROGÈNE ET EXTENSIBLE POUR SPACE

Ce chapitre décrit la méthodologie utilisée pour consolider l'environnement SPACE autour de la plateforme virtuelle actuelle en **(1)** étendant sa conformité au standard CoreConnect [7] d'IBM et en **(2)** intégrant un ISS du PowerPC405Fx tel que retrouvé sur les FPGA Virtex4 de la compagnie Xilinx.

Le travail consiste donc à passer des spécifications des composants de la plateforme CoreConnect à des modèles de simulation TLM et à intégrer un ISS du PowerPC405 dans SpaceLib selon les concepts de co-conception (ANNEXE A). Un environnement de développement C/C++/SystemC (ANNEXE B) sous Eclipse ainsi que SpaceStudio ont été utilisés comme outils de conception.

3.1. La plateforme CoreConnect d'IBM implémentée par Xilinx

IBM a proposé une spécification de protocole générique de bus nommé CoreConnect qui fut implémentée par Xilinx pour le FPGA Virtex4. Ce protocole est une organisation hiérarchique de trois bus :

- Un bus à haut débit pour les périphériques rapides et les processeurs nommé PLB [54] (*Processsor Local Bus*) ;
- Un bus à bas débit pour les périphériques lents nommé OPB [55] (*On-Chip Peripheral Bus*) ;
- Un bus de contrôle et de diagnostique nommé DCR [56] (*Data Control Register*).

3.1.1. Le bus PLB

Le PLB répond aux requis de faible latence et de débit élevé en offrant les caractéristiques suivantes :

- Des bus de données de 64 bits ;
- Un bus d'écriture et de lecture séparés et indépendant permettant la concurrence de transferts ;
- Transferts en mode rafale, DMA, scindée (« split transaction ») ou ligne de cache ;
- Un pipeline permettant le chevauchement de transactions non complétées.

3.1.2. Le bus OPB

Le bus OPB a pour objectif de réduire l'utilisation du bus PLB en y reléguant les périphériques plus lents e.g. minuteries, UARTs, etc. Il offre les caractéristiques suivantes :

- Un bus de données de 32 bits ;
- Un redimensionnent dynamique pour les transferts de 8 et 16 bits ;
- Un mode séquentiel équivalent à un mode rafale ;
- Un signal *bus lock* pour conserver l'accès au bus.

3.1.3. Le bus FCB

Le FCB (ou *Fabric Coprocessor Bus*) permet de relier le PowerPC405FX à plusieurs coprocesseurs via un composant nommé APU, ou *Auxiliary Processor Unit*, en utilisant des instructions spécialisées. Certaines instructions prédéfinies de l'APU permettent la

lecture/écriture de données via le bus FCB. Le lien est bidirectionnel et le PowerPC possède une seule interface maîtresse.

3.1.4. Le bus FSL

Le FSL [57] (ou *Fast Simplex Link*) est un lien point-à-point développé par Xilinx qui permet de relier le processeur embarqué μBlaze [58] (μB) à un module spécialisé. Ce lien est unidirectionnel. Il faut instancier 2 FSL pour créer une communication bidirectionnelle. Le μB peut instancier 8 FSL maîtres et 8 FSL esclaves. Les données sont tamponnées dans une FIFO à profondeur configurable et permet la communication bloquante ou non bloquante.

3.2. Le PowerPC405FX

Le PowerPC405 est une addition à la famille de processeurs embarqués RISC PowerPC 400. Son cœur occupe une petite surface (2.0 mm^2 avec technologie 0.25 μm), consomme peu d'énergie (400 mW @ 200 MHz), est versatile et compatible avec la plateforme PowerPC. Il s'intègre à l'architecture de communication CoreConnect [59].

3.2.1. Architecture

Le PowerPC405 possède de nombreuses interfaces dont deux interfaces PLB 64-bits pour les données et les instructions, une interface DCR pour la configuration de composants externes et la réduction du trafic PLB, deux interfaces OCM dont la latence d'accès équivaut à un succès de cache (*cache hit*) pour les instructions et données (Figure 3.1). En plus de ces interfaces de bus, le PowerPC405 possède une interface APU pour étendre son jeu d'instructions via un processeur auxiliaire (ANNEXE E). Il possède aussi une interface EIC *(External Interrupt Controller)* qui étend le support des

interruptions pour la logique externe. Cette interface fournit deux niveaux de priorités pour les interruptions : (1) critique et (2) externe.

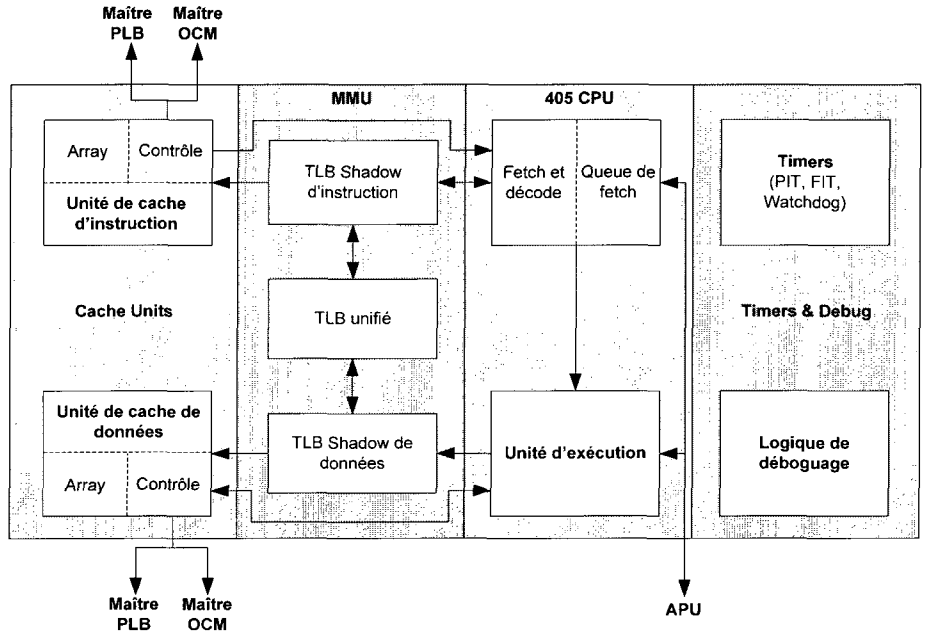


Figure 3.1 – Architecture simplifiée du PowerPC405

Le PowerPC 405 possède un pipeline 5 étages (*fetch, decode, execute, writeback* et *load write-back*). Pour obtenir un flot constant d'instruction dans le pipeline, la logique de *fetch* alimente une queue (FIFO) d'instructions d'une profondeur de 3. La prédiction de branchement est statique et suit quelques règles de base détaillées en [59].

Au total, le PowerPC405 peut gérer 19 exceptions et interruptions générées de manière interne (minuteries, événement de débogage, etc.) ou via l'EIC. Elles sont divisées en deux classes : (1) critiques et (2) non critiques. Chaque classe a sa propre paire de registres de sauvegarde pour contenir le statut du processeur. Ceci accélère le traitement des interruptions critiques.

Le cœur accède à la mémoire via les unités de cache possédant chacune une interface PLB et OCM. Les succès de cache sont considérés comme des accès mémoire en 1 cycle tandis que les défauts de cache reviennent à un accès bus vers une mémoire externe. Les

caches sont non bloquantes, permettant au PowerPC de simultanément exécuter des instructions et en quérir de nouvelles sur le bus. Le pipeline n'est donc jamais interrompu.

L'unité de contrôle de la mémoire (MMU ou *Memory Management Unit*) supporte des tailles de pages multiple ainsi qu'une variété d'attributs de protection de stockage et d'options de contrôle d'accès (*cacheability, write through/write back mode, allocate on write, big/little endian, guarded* et *user-defined*). Le MMU fournit le support de la mémoire virtuelle par translation d'adresses via une TLB à 64 entrées. Le mode virtuel peut être désactivé.

Le cœur comporte trois minuteries : le PIT (intervalles programmables), le FIT (intervalles fixes) et un Watchdog. Le compteur de base est de 64 bits et est incrémenté à chaque coup d'horloge. Des interruptions sont générées lorsque les minuteries arrivent à expiration.

3.2.2. Les composants APU et FCB

L'APU permet au concepteur d'étendre le jeu d'instructions natif du PowerPC405 avec des instructions personnalisées qui sont exécutées dans des modules coprocesseurs (*Fabric Coprocessor Module*, ou FCM) sur le FPGA. Ce mécanisme permet une intégration beaucoup plus forte d'une fonction spécifique au pipeline du processeur que ne le permet un périphérique bus [60]. La Figure 3.2 présente le flot de pipeline entre le PowerPC405, le contrôleur APU et un FCM. Le contrôleur APU permet deux fonctions :

- Effectuer la synchronisation de domaines d'horloge entre le PowerPC405 et le FCM ;
- Décoder certaines instructions FCM et notifier le processeur des ressources requises par l'instruction, e.g. les sources de données dans les registres.

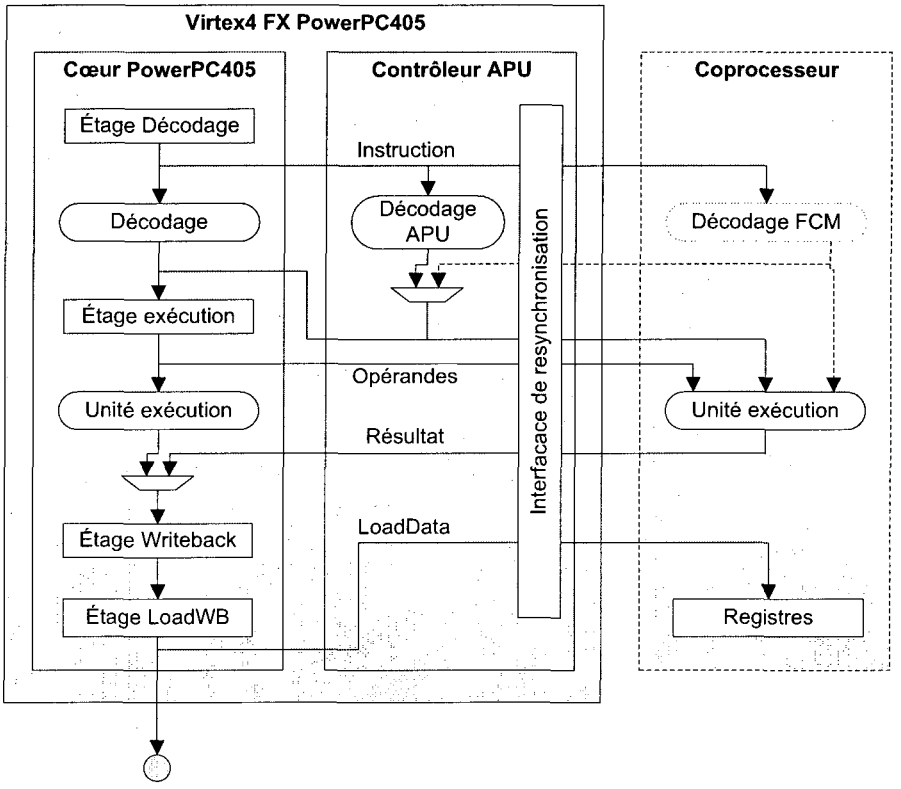


Figure 3.2 – Relation entre le contrôleur APU et le pipeline du PowerPC405

3.2.3. Instructions spécialisées

Dépendamment de l'application FCM, le contrôleur APU peut décoder toutes les instructions, aucune instruction ou une portion des instructions pendant que le FCM décode les autres. Un **décodage par l'APU** signifie que l'APU détermine les ressources du processeur requises pour l'exécution de l'instruction et donne cette information au CPU. Le FCM peut aussi effectuer cette partie du décodage et passer l'information au contrôleur APU. Cependant, pour déterminer la fonction complète d'une instruction, le FCM doit décoder entièrement l'instruction et supporter son exécution en entier. Deux types d'instruction peuvent être supportés par un FCM :

- **Définie par l'utilisateur (UDI) :** Une instruction dont le format est configurable et constitue une véritable extension de l'ISA du PowerPC. Huit instructions de ce type peuvent être définies et configurées à l'aide de certains registres de l'APU.
- **Prédéfinie :** Une instruction dont le format est définie par le jeu d'instruction du PowerPC et où le FCM devient un simple coprocesseur performant une exécution définie par l'ISA [61].

Les extensions de l'ISA sont définies par leur interaction avec l'exécution normale du pipeline du processeur. Ceci mène à trois classes d'instructions :

- **Instructions autonomes :** Elles ne bloquent pas l'exécution du pipeline. Elles sont typiquement du type *fire-and-forget* et ne retournent pas d'état ou de données au processeur. Elles peuvent bloquer le pipeline si une exécution n'est pas complétée lors d'un appel subséquent à la même instruction.
- **Instructions non autonomes :** Elles bloquent l'exécution du pipeline jusqu'à la complétion de l'instruction. Elles retournent typiquement un état ou une donnée au processeur.
    - **Instructions bloquantes :** Une instruction ne pouvant pas être annulée de manière prévisible et ré-exécutée ultérieurement. Une fois que l'instruction a complétée le cycle *Execute*, les interruptions et exceptions sont désactivées afin d'assurer la complétion de l'exécution.
    - **Instructions non bloquantes :** Une instruction pouvant être annulée de manière prévisible et ré-exécutée ultérieurement.

3.3. Intégration de l'ISS du PowerPC à SpaceLib

Afin de fournir une plateforme virtuelle multiprocesseur complète dans SPACE permettant le développement du DirectLink, présenté au chapitre 4, il est nécessaire de

développer un ISS du PowerPC405 pour SpaceLib. Le code source du programme PSIM a été utilisé afin de minimiser le temps de développement.

3.3.1. Le simulateur PSIM

PSIM est un programme écrit en ANSI-C qui émule l'ISA de la famille de microprocesseur PowerPC. Le code source est disponible gratuitement sous licence GNU et inclus dans GDB, le débogueur GNU [62]. PSIM respecte les trois niveaux de conformité de l'architecture PowerPC définis dans [63]:

- UEA (*User Environment Architecture*) : les registres, instructions, modèle de données et d'exécution offert à toutes les applications.
- VEA (*Virtual Environment Architecture*) : les caractéristiques de l'architecture qui permettent aux programmes de créer ou modifier du code, partager des données dans un système multiprocesseur et optimiser les performances des accès données.
- OEA (*Operating Environment Architecture*) : les caractéristiques de l'architecture qui permettent au système d'exploitation d'allouer et gérer le stockage de données, supporter les exceptions et les périphériques d'entrée-sortie et offrir les services de sécurité propres aux OS multiprocesseurs modernes.

Les modèles de PowerPC supportés par PSIM sont les 601, 603 et 604 qui respectent la spécification du BOOK-E [64] mais pas celle de l'*IBM PowerPC Embedded Environment* [65].

3.3.1.1.Architecture de PSIM

PSIM est un énorme programme comportant plusieurs dizaines de milliers de lignes de code. La Figure 3.3 en définit sommairement l'architecture.

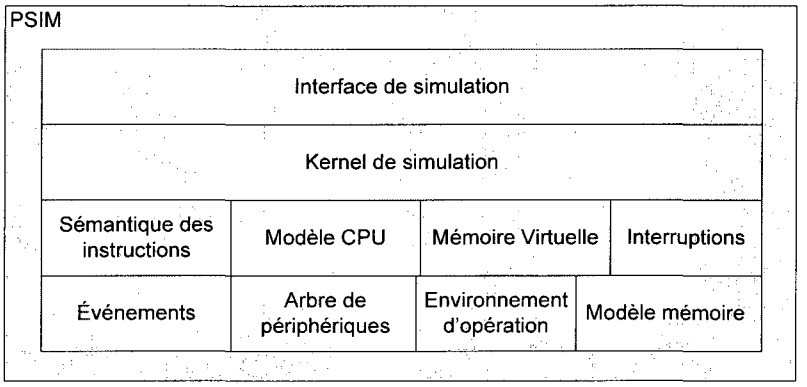


Figure 3.3 – Schématisation de l'architecture de PSIM

PSIM est un ensemble de fonctions qui manipulent un gigantesque agrégat de structures représentant le processeur PowerPC. PSIM fournit une interface de simulation qui permet d'effectuer certaines opérations sur une simulation de programme telles le lancement, l'arrêt, le redémarrage et l'exécution pas à pas. Le *kernel* de simulation comprend la boucle d'exécution principale, une cache d'instructions pré-décodées pour accélérer la simulation et un arbre de tableaux de pointeurs de fonctions utilisé pour décoder les instructions. Chaque feuille de l'arbre pointe vers une fonction représentant la sémantique d'une instruction.

Lorsqu'une fonction sémantique est appelée, elle manipule un modèle du processeur comportant les registres, l'état du processeur, les minuteries, etc. S'il y a des accès mémoire, un ensemble de fonctions modélise le comportement de la mémoire virtuelle pour effectuer la translation d'adresse et la gestion du TLB. Au même niveau, un ensemble de fonction modélise le comportement des interruptions sur le modèle du CPU.

Le tout se base sur un mécanisme d'événements, un arbre de périphériques, un modèle d'environnement d'opération et un modèle de la mémoire. Le mécanisme d'événement est une queue prioritaire (*priority queue*) de structures représentant des événements et qui permet d'ordonnancer l'exécution de fonctions comme les interruptions, les exceptions, les minuteries, etc. Le modèle d'environnement d'opération permet de simuler le comportement des appels systèmes d'applications compilées pour Solaris,

Linux et OpenBSD au niveau UEA, donc s'exécutant dans un environnement virtuel (ou en mode utilisateur).

L'arbre de périphérique fournit une représentation abstraite d'une architecture complète d'ordinateur. Cette approche propose un *framework* dans lequel une implémentation de système informatique peut être modelée. La modélisation peut comprendre un disque dur, des transferts DMA, des registres de périphériques, un réseau d'interruption, un contrôleur d'interruption, etc. Cet arbre est séparé du kernel de simulation mais interagit avec celui-ci.

Le modèle de la mémoire stocke quant à lui tous les segments d'une application binaire compilée pour le PowerPC dans la mémoire virtuelle de PSIM. Certaines fonctions (lecture, écriture) sont fournies pour manipuler la mémoire en prenant en compte des attributs d'accès. Le modèle gère aussi la carte de mémoire du système (*memory map*), créée lors de l'initialisation de la simulation à partir des informations stockées dans l'arbre de périphérique. Lorsque le processeur lit ou écrit en mémoire, la requête passe directement par le nœud de l'arbre capable de gérer l'accès mémoire spécifique à une adresse.

3.3.1.2.Modifications apportées à PSIM

Les architectures des PowerPC 60X et 405 sont compatibles au niveau UEA mais diffèrent au niveau VEA et OEA qui sont optimisées pour rencontrer les requis des applications embarquées. Ces optimisations incluent la gestion de la mémoire, de la cache, les exceptions, les minuteries [66]. Ces optimisations sont reflétées par les registres spécialisés et les instructions supportées. Puisque les applications SPACE s'exécutent en mode noyau (OEA), ces modifications doivent être apportées à PSIM. Voici la liste des ajouts effectués :

- **Instructions :** Onze instructions (Tableau 3.1) ont dues être implémentées. Ceci s'effectue par la création de fonctions *sémantiques* telles qu'illustrées dans l'exemple suivant. Un pointeur de fonction est ensuite stocké dans le tableau de décodage à l'index correspondant au code d'instruction primaire. Cette méthode permet un décodage d'instruction en quelques cycles, accélérant la cosimulation.

Tableau 3.1 – Instructions ajoutées à PSIM

Instruction	Niveau architectural	Description
iccci / dccci	OEA	Invalider une ligne de cache d'instruction / de données
icread /dcread	OEA	Lire une ligne de cache d'instruction / de données
eieio / isync	OEA	Forcer une synchronisation de contexte
rfci	OEA	Retour d'une interruption critique
wrtee / wrteei	OEA	Activation / Désactivation des interruptions externes
mtdcr / mfdcr	OEA	Lecture / Écriture via le bus DCR

```
unsigned_word semantic_Write_External_Enable_Immediate (...)
{
  NextInstructionAddr = CurrentInstructionAddr + 4;

  if (IS_KERNEL_MODE())
  {
      program_interrupt();
  }
  else
  {
      if (Instruction & EE_MASK) MSR |= msr_external_interrupt_enable;
      else MSR &= ~msr_external_interrupt_enable;
      check_masked_interrupts(processor);
  }

  return nia;
}
```

Figure 3.4 – Exemple de sémantique d'instruction

- **Minuteries :** Le PIT (*Programmable Interval Timer*) remplace le *decrementer* du PowerPC6XX. Le FIT et le watchdog ont été ajoutés.

- **Interruptions :** Le vecteur d'exception et d'interruption est complètement modifié (Tableau 3.2) par la création de nouvelles fonctions simulant leur comportement (modifications apportées à l'état du processeur).

Tableau 3.2 – Modifications au vecteur d'interruption

Offset	PowerPC405	PowerPC6XX
0x0100	Critical-Input	System Reset
0x0900		Decrementer
0x0D00		Trace
0x0F00		Performance Monitor
0x0F20	APU Unavailable	
0x1000	PIT	Instruction translation miss
0x1010	FIT	
0x1020	Watchdog	
0x1100	Data-TLB Miss	Data translation miss (load)
0x1200	Instruction-TLB Miss	Data translation miss (store)
0x1300		Instruction-address breakpoint
0x1400		System Management
0x1700		Thermal Management
0x2000	Debug	

- **Registres :** L'étendue des modifications apportées au niveau des registres est majeure et profonde. Le lecteur devrait se reporter à [66] pour constater les différences entre les deux familles de PowerPC. Les nouveaux registres sont représentés par des variables *unsigned long* stockées dans la structure modélisant l'état du processeur.

Certaines fonctionnalités de PSIM ont été retirées :

- **Arbre de périphériques et modèle de mémoire :** Ces deux mécanismes de PSIM ne sont pas en harmonie avec la philosophie de SPACE car non simulable

au niveau transactionnel. Ils ont été remplacés par un mécanisme de callback expliqué à la section suivante.

- **Environnement d'exécution :** Ce concept est inutile puisque toutes les applications SPACE roulent en mode *kernel* et sur des RTOS non supportés par le modèle PSIM.
- **Mémoire virtuelle :** La gestion de la mémoire virtuelle ne s'effectue pas de la même manière entre les deux familles de PowerPC. De plus, toutes les fonctions effectuant la translation d'adresse ont été déplacées dans un autre module SystemC pour réduire le couplage et augmenter la modularité de l'ISS.

Finalement, afin d'accélérer la simulation et minimiser la complexité du code, la boucle de décodage a été intégrée directement à l'interface de simulation.

3.3.2. Méthodologie d'intégration et interface avec SPACE

Une fois PSIM modifié pour modéliser adéquatement l'ISA du PowerPC405Fx, il faut l'intégrer à un module SystemC modélisant le comportement des différents composants du processeur expliqués en 3.2.

Dans la méthodologie proposée, contrairement à celle détaillée dans [24], le code de PSIM ne s'exécute pas dans un processus indépendant de la simulation de la plateforme matérielle. Afin d'éviter l'implantation de mécanismes de synchronisation interprocessus et d'éliminer les délais induits par la communication par socket, l'ISS est intégré à un modèle SystemC du processeur. Ce modèle comprend plusieurs modules SystemC dont une unité de gestion de la mémoire et des caches, un serveur GDB et des contrôleurs d'interface.

L'agrégat de structures représentant le modèle de simulation de PSIM devient un attribut de classe d'un module SystemC. L'ensemble des fonctions de PSIM est compilé dans

une librairie statique C qui est liée à l'exécutable de simulation SPACE. Il est toujours possible d'interagir avec l'ISS de l'extérieur et de déboguer l'application à l'aide de GNU GDB via une communication interprocessus de type socket.

3.3.2.1.Intégration de PSIM dans un modèle SystemC

Une fois PSIM déclaré comme attribut privé du module SystemC du PowerPC405, il faut implémenter un moyen pour que PSIM puissent accéder à la mémoire, aux périphériques et aux interfaces qui sont modélisés par d'autres modules SystemC dans SPACE. Ceci s'effectue par un remplacement de l'arbre de périphériques de PSIM par des structures de fonctions *callbacks* pour chaque interface physique disponible soit le DCR, l'APU et la mémoire (via le PLB et l'OCM). Ces fonctions *callbacks* manipulent les classes de contrôleur d'interfaces (Figure 3.5).

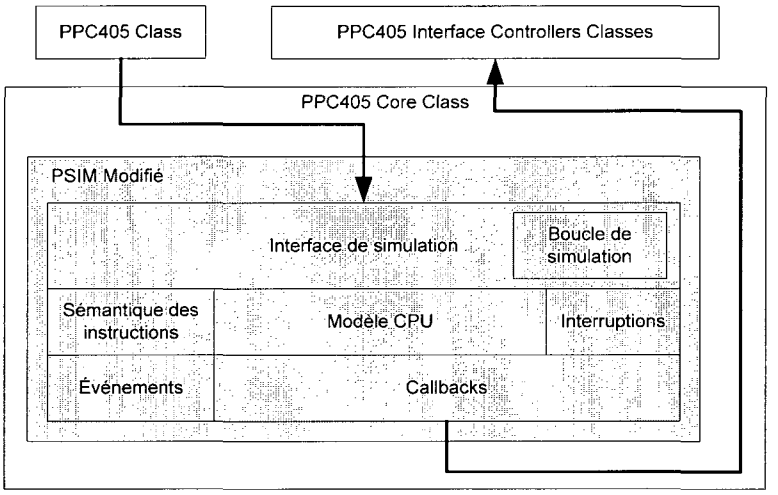


Figure 3.5 – Schématisation de l'intégration de PSIM à SPACE

Cette méthode permet une plus grande flexibilité et réutilisation de PSIM. Il est possible de changer le comportement des interfaces, de simuler plusieurs niveaux de granularité temporelle ou de changer les paramètres des caches de manière non invasive du point de vue de PSIM.

L'exemple ci-dessous illustre comment cette méthode est implantée. Une structure est définie pour chaque interface que possède le PowerPC, e.g. la mémoire (ou MMU). Ces structures sont intégrées au modèle du processeur. Elles contiennent un pointeur de fonction pour chaque opération primitive que le PowerPC peut vouloir effectuer sur une interface, e.g. une synchronisation de contexte, une lecture de mot de 32 bits, etc. Elle contient aussi un pointeur vers l'instance de la classe qui s'occupe de gérer l'interface.

```
typedef int (*FP_CALLBACK_MEM_ACCESS)( int *class_ptr,
unsigned long address, unsigned long *data);

typedef int (*FP_CALLBACK_OP)( int *class_ptr);

typedef struct _mmu_callbacks
{
  int *class_ptr;

  FP_CALLBACK_MEM_ACCESS read_4;
  FP_CALLBACK_MEM_ACCESS write_4;
  FP_CALLBACK_MEM_ACCESS read_2;
  FP_CALLBACK_MEM_ACCESS write_2;
  FP_CALLBACK_MEM_ACCESS read_1;
  FP_CALLBACK_MEM_ACCESS write_1;

  FP_CALLBACK_MEM_ACCESS fetch;

  FP_CALLBACK_OP sync_context;
  FP_CALLBACK_OP tlb_invalidate_all;
  FP_CALLBACK_OP tlb_invalidate_entry;
  FP_CALLBACK_OP translate_ea;

} mmu_callbacks;


struct _cpu {
        …
        mmu_callbacks memory;
        apu_callbacks apu;
        dcr_callbacks dcr;
        …
};
```

Figure 3.6 – Exemple de structure de callback

L'exemple suivant illustre une fonction de registration. Elle permet au niveau supérieur (ppc405Core) de fournir les fonctions à utiliser par PSIM pour accéder aux interfaces, e.g. écriture et lecteur via le bus DCR. Elle indique aussi quelle instance de classe s'occupera de traiter les requêtes via un pointeur. L'étape de registration s'effectue pendant la phase *end_of_elaboration* de SystemC, i.e. le moment où SystemC finalise la création de ses modules.

```
void cpu_bind_dcr
(cpu *processor, int * pClass, FP_CALLBACK_MEM_ACCESS pDCRWrite,
 FP_CALLBACK_MEM_ACCESS pDCRRead)
 )
{
        processor->dcr.class_ptr = pClass;
        processor->dcr.write = pDCRWrite;
        processor->dcr.read = pDCRRead;
}

void ppc405Core::end_of_elaboration()
{
        …
        //register dcr callbacks
        cpu_bind_dcr (
                psim_cpu (simulation),
                (int*)pDCRController,
                DCRWrite,
                DCRRead
        );
        …
}
```

Figure 3.7 – Exemple de fonction de registration

Il est impossible de passer une fonction membre de classe comme fonction de callback, puisque PSIM ne connait pas les définitions des classes du modèle. Une fonction callback est donc implantée avec une fonction statique non-membre mais déclarée dans le contexte (fichier .cpp) de la classe ppc405Core. C'est ici qu'intervient le pointeur vers la classe gérant une interface. Il est passé en argument à la fonction callback lors de son appel pour qu'elle puisse ensuite appeler les méthodes appropriées, e.g. la synchronisation du MMU.

```
static int SyncContext(int *class_ptr)
{
        unsigned long msr, ccr0, ccr1, iccr;
        ((ppc405Core*)(class_ptr))->readRegister("msr", msr);
        ((ppc405Core*)(class_ptr))->readRegister("ccr0", ccr0);
        ((ppc405Core*)(class_ptr))->readRegister("ccr1", ccr1);
        ((ppc405Core*)(class_ptr))->readRegister("iccr", iccr);

        ((ppc405Core*)(class_ptr))->pMMU->syncWithCore(msr, ccr0, ccr1, iccr);

        return 0;
}

//Utilisation
processor->memory.sync_context (processor->memory.class_ptr);
```

Figure 3.8 – Exemple de fonction callback et utilisation

3.3.2.2.Architecture modulaire

Le modèle SystemC résultant de cette intégration de PSIM est une agrégation de classes (Figure 3.9). La façade du modèle est la classe **PPC405** qui est instanciée dans une architecture SPACE et qui comporte tous les ports SystemC requis pour connecter les bus et les signaux d'interruption. C'est également un *Proxy* pour la classe **PPC405Services** qui possède une instance de toutes les classes de contrôleurs d'interfaces, du cœur et du MMU, ainsi que les méthodes SystemC utilisées pour gérer les événements des interruptions. L'utilisation d'un *Proxy* est nécessaire pour cacher au client les détails d'implantation du modèle.

La classe **GDBServer** permet à un client GDB (débogueur GNU) de se connecter à la simulation via une connexion par socket. Il est ainsi possible de lire le contenu des registres, de la mémoire, d'insérer des *breakpoints* et de contrôler la simulation du processeur.

C'est la classe **ppc405Core** qui encapsule PSIM, fournit les méthodes de contrôle de la simulation et initialise le modèle PSIM avec les fonctions callback appropriées. Pour ce faire, elle contient des pointeurs vers chaque classe de contrôleur d'interface. Ces dernières sont nombreuses (**DCRController, OCMController, PLBController, APUController**) et fournissent une abstraction du comportement des interfaces de communication du processeur.

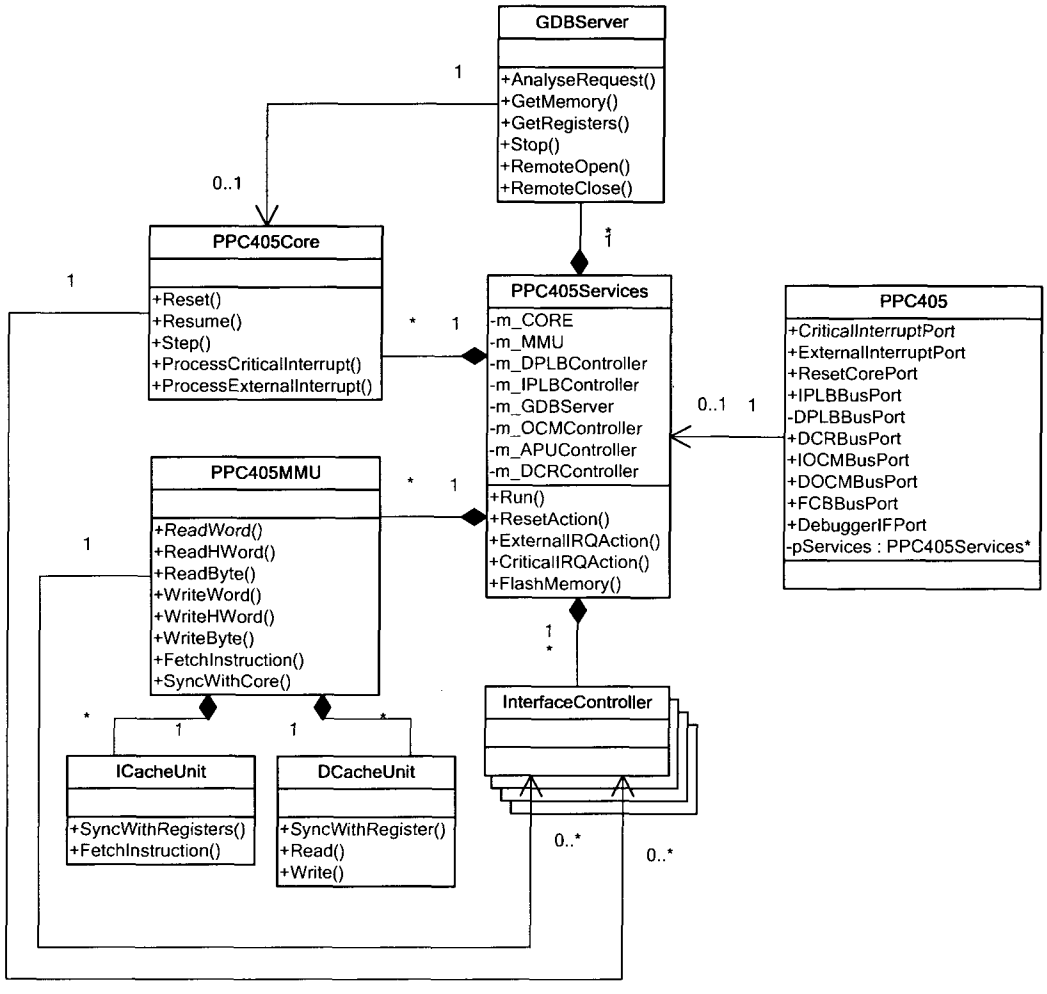


Figure 3.9 – Diagramme de classes simplifié du PowerPC405

La classe **PPC405MMU** implémente les stratégies d'accès mémoire, le support de la mémoire virtuelle par translation d'adresse et les caches du processeur. C'est elle que toutes les fonctions callback de mémoire manipulent pour lire, écrire des données ou obtenir une instruction. Elle contient un processus léger, ou fil d'exécution, indépendant qui alimente sans cesse en instructions le cœur de simulation en les stockant dans une FIFO selon le modèle décrit à la section 3.2.1.

Le cœur de simulation de l'ISS résultant correspond au niveau TF de la méthodologie SystemC puisque le pipeline du processeur n'est pas implémenté. Cependant, les interfaces I/O du processeur sont implémentées au niveau BCA.

3.3.2.3.Port du RTOS

L'ajout de ce nouveau simulateur du PowerPC405 dans SPACE nécessite le port du système d'exploitation temps réel µC/OS-II et des couches de communication bas niveau du Tor, e.g. la pile logicielle de SPACE.

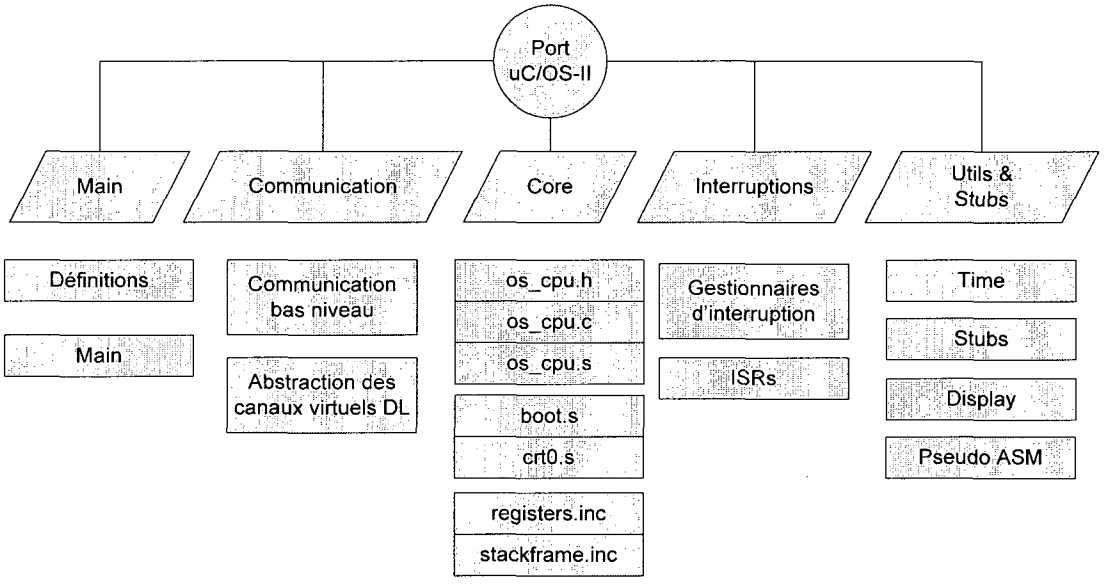


Figure 3.10 – Structure du port d'µC/OS-II pour le PowerPC405

Le port effectué pour le PowerPC405 (Figure 3.10) est constitué de 5 parties :

- **Main :** Fonction *main* du logiciel applicatif – initialise l'environnement logiciel SPACE, lance les tâches utilisateur, etc. – et définitions de certains paramètres architecturaux.
- **Communication :** Fonctions d'abstraction des communications à bas niveau pour le bus PLB et du DirectLink (voir chapitre suivant).

- **Core :** Cœur du RTOS. Comprend les fonctions d'initialisation de tâches, de routines de changement de contexte, d'initialisation de l'environnement C++, de démarrage, etc.
- **Interruptions :** Définition des routines de service d'interruption pour l'adaptateur de communication SPACE, des minuteries et des interruptions externes.
- **Utils & Stubs :** Ensemble de fonctions utilitaires pour le calcul du temps, l'affichage et de pseudo assembleur.

3.3.2.4.Plateforme dans SPACE

Ce nouveau modèle de processeur dans SPACE ouvre la porte à de nouvelles possibilités de conception de plateforme, i.e. architecture de communication multibus hiérarchique, multiprocesseur hétérogène, extensibilité du jeu d'instruction, etc. Le mécanisme de communication SPACE induit toutefois des prérequis architecturaux tels la présence de nouveaux composants e.g. un contrôleur d'interruption (PIC) et un ISSAdapter pour le PLB. La présence de deux niveaux d'interruption externes sur le PowerPC405 permet l'accélération du traitement des communications matérielles vers logiciel dans SPACE.

La section interruption du port de µC/OS-II définit deux routines d'interruption :

- **Critique** : Routine spécifique à la gestion des messages SPACE provenant de la partition matérielle. L'interruption de l'ISSAdapter y est connectée.
- **Non critique** : Routine générique pour toutes les interruptions externes pouvant provenir de modules de la partition matérielle. Le concepteur peut registrer des fonctions de gestion spécifique à chaque interruption. L'interruption du PIC y est connectée. Une boucle de traitement analyse quelles interruptions doivent être traitées.

Ces deux niveaux étaient auparavant combinés, induisant le délai de la boucle de traitement lors d'interruption causée par une communication matériel/logiciel. Cette séparation permet de réduire le délai d'interruption prétraitement qui est non négligeable comme l'a démontré [67].

3.3.3. Modélisation de l'APU et du FCB

Une section importante du PowerPC405Fx disponible sur le Virtex4 de Xilinx est l'APU et le bus FCB. Cette section décrit l'architecture du modèle SPACE et son implémentation. Ces composants seront utilisés au chapitre 4.

3.3.3.1.Architecture

Selon la spécification du FCB, ce bus est un ensemble de fils, sans logique de contrôle, qui envoie chaque signal de l'APU à tous les FCM connectés au bus. Chaque instruction qui étend l'ISA du PowerPC via l'APU ne peut être décodé et exécutée que par **un seul et unique** FCM. Quand une instruction est mise sur le bus, chaque FCM est responsable de vérifier s'il en est le récipiendaire et effectuer le traitement approprié si tel est le cas. Ce comportement est réalisable en simulation mais consommerait trop de temps : il faudrait envoyer une requête à chaque FCM pour vérifier lequel peut exécuter une instruction spécifique. Pour contourner ce problème, l'architecture illustrée à la Figure 3.11 a été adoptée.

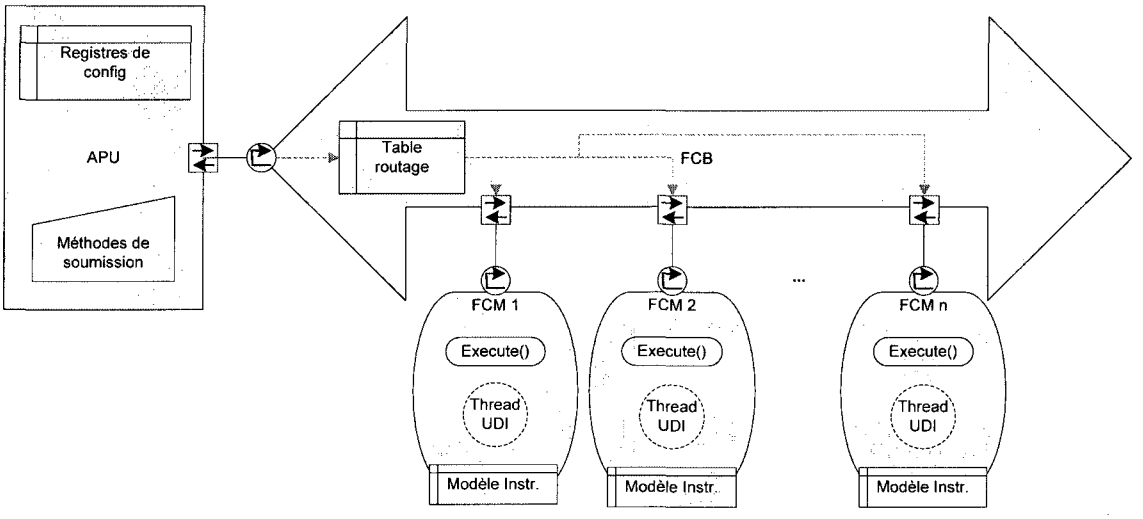


Figure 3.11 – Architecture de l'APU et du FCB

Dans cette architecture, l'APU est un module SystemC intégré à l'ISS du PowerPC405. L'APU fournit diverses méthodes pour lancer l'exécution d'une instruction. Il en existe cinq soit une pour chaque classe possible : UDI et prédéfinies (opération FPU, accès mémoire FPU, *load* FCM et *store* FCM). Les méthodes pour instructions prédéfinies sont appelées directement par les fonctions sémantiques d'instruction. Dans le cas où une instruction n'est pas dans le tableau de décodage du PowerPC405, la méthode UDI est appelée si le format de l'instruction correspond à celui spécifié par un des huit registres de configuration UDI de l'APU. Ces cinq méthodes formatent une transaction et la dépose sur le bus FCB qui se charge d'effectuer le routage vers le bon FCM.

Pour effectuer le routage, le FCB possède une table comportant une entrée pour chaque instruction pouvant être décodée par l'ensemble des FCM. Cette table associe un numéro de port SystemC à chaque instruction et est construite lors de la phase de fin d'élaboration de la simulation SystemC. Chaque FCM est responsable de fournir un modèle d'instruction pour chaque instruction qu'il décode. Un modèle comprend un Masque et un Template. Le routage s'effectue sur le code d'opération d'une instruction si :

$$CodeOp \,\&\, Masque = Template$$

Cette méthode permet de facilement associer un sous-ensemble d'instruction à un FCM spécifique. De plus, chaque FCM comprend un *thread* qui peut être lancé lors d'instructions autonomes.

3.3.3.2.Implémentation

Cette description architecturale a été implémentée selon le diagramme de classe illustré à la Figure 3.12. La classe **APUUtility** permet de manipuler les instructions pour en extraire des informations, i.e. registre d'opérandes, code d'opération, etc., ou générer les modèles d'instruction pour le routage. Le patron de conception Proxy est ici aussi utilisé pour cacher l'implémentation des modules d'une simulation SPACE.

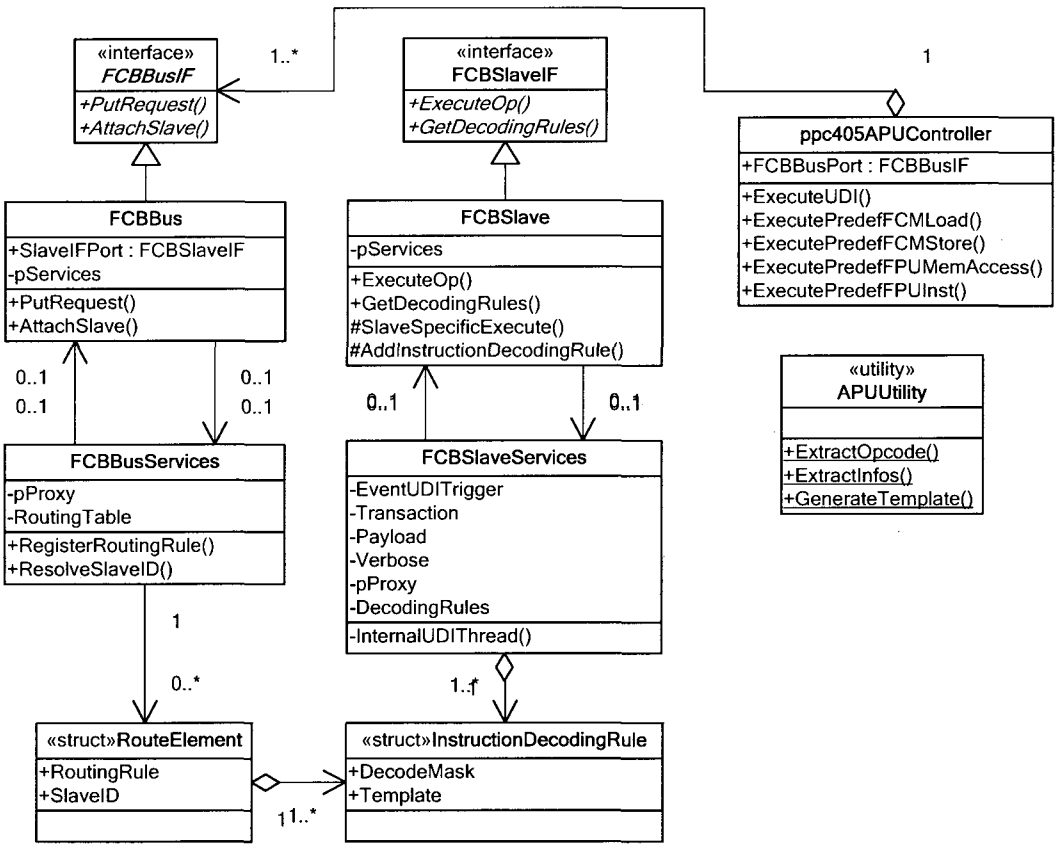


Figure 3.12 – Diagramme de classe simplifié de l'APU

Un esclave FCB (classe **FCBSlave**, représentant un FCM) implémente une interface spécifique **FCBSlaveIF** comportant une méthode pour exécuter une instruction et une méthode pour récupérer les modèles d'instructions. Un module SPACE désirant se connecter au FCB doit hériter de FCBSlave et implémenter une fonction virtuelle pure (**SlaveSpecificExecute**) modélisant le comportement des instructions dont les modèles sont fournis par la fonction **AddInstructionDecodingRule**.

Le bus FCB (classe **FCBBus**) reçoit les instructions via la méthode **PutRequest()** et les route au bon esclave.

3.4. Un environnement multiprocesseur hétérogène

Cet environnement de simulation consolidé autour de la technologie disponible sur les FPGA Virtex-4 de Xilinx est multiprocesseur et hétérogène, en ce sens qu'elle offre à la disposition du concepteur des processeurs de types différents tels le MicroBlaze et le PowerPC405 FX. De ce fait, les requis d'extensibilité des jeux d'instructions et de la présence d'interfaces pour canaux à haut débit sont remplis par les modèles de simulation intégrés à SPACE afin de développer de nouveaux mécanismes de communication point-à-point.

CHAPITRE 4

DIRECTLINK : ABSTRACTION DES COMMUNICATIONS POINT-À-POINT DANS LA PLATEFORME VIRTUELLE SPACE

4.1. Paradigme du DirectLink

Le paradigme DirectLink [73] consiste en un mécanisme de communication point à point pouvant s'adapter aux architectures MPSoC hétérogènes, peu importe la plateforme matérielle utilisée. Il considère les modèles de communication de la section 2.1, excluant le modèle logiciel/logiciel puisque les mécanismes actuels de SPACE le gèrent déjà de manière point à point.

DirectLink tire avantage, s'il y a lieu, de la présence de canaux de communication dédiés ou de la possibilité d'étendre le jeu d'instruction du processeur (ANNEXE E). L'extensibilité du processeur est ici utilisée pour améliorer la bande passante d'une architecture et non pas sa puissance de calcul brute. Le canal partagé est désengorgé par la création de chemins de données spécifiques.

DirectLink se présente sous la forme d'une abstraction complète des implémentations possibles, tant matérielles que logicielles. De plus, du point de vue programmeur, une communication DirectLink se fait via les mêmes primitives définies en 2.3.1.2, soit ModuleRead et ModuleWrite. Une sous-couche de l'implémentation se charge d'effectuer le routage des paquets entre les liens point à point et le canal partagé. De cette manière, la communication entre deux modules peut se faire même si un lien point à point est retiré.

Les requis fonctionnels, architecturaux et de performances sont les suivants :

- Modéliser la transmission de messages ;
- Réduire l'*overhead* induit par le logiciel SPACE;
- Supporter les communications bloquantes et non bloquantes ;
- Améliorer la latence et le débit des communications ;
- Aider à diminuer la consommation de puissance (section 2.2.3) par le support de domaines d'horloge multiples ou la réduction de la complexité ;
- Fournir un modèle architectural générique au développeur ;
- Être configurable (profondeur des FIFOs, sélection d'un canal virtuel, etc.) ;
- Offrir une interface unique, tant matérielle que logicielle ;
- Abstraire les spécificités de la plateforme telles l'architecture de communication, les protocoles utilisés, les instructions spécialisées, etc. ;
- Se concilier à la souplesse du *drag-and-drop* SPACE.

4.2. Méthodologie

L'implémentation du paradigme DirectLink se fait selon le choix de l'architecture cible qui impose certains protocoles de communication auxquels les modules matériels d'un système et les tâches logicielles devront être adaptés. L'architecture cible est ici la plateforme multiprocesseur hétérogène décrite au chapitre 3 basée sur la technologie du Virtex4 de Xilinx. Le chapitre 5 expliquera comment le paradigme peut être étendu à d'autres architectures.

La première étape de l'implémentation consiste en la réalisation des nouvelles interfaces de communication entre les différents composants d'un système. Ces interfaces capturent les fonctionnalités requises et mentionnées à la section précédente. La deuxième étape est la réalisation d'une abstraction du lien point-à-point s'intégrant facilement à SPACE. Cette abstraction est à la fois une classe pouvant s'instancier dans

une simulation SystemC/SPACE et un mode de visualisation des liens point à point dans SpaceStudio.

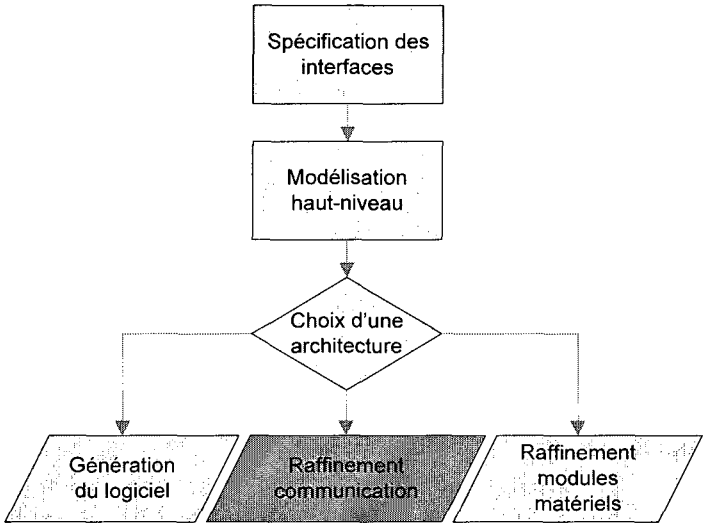


Figure 4.1 – Méthodologie d'implémentation du DirectLink à une architecture cible

L'étape finale est le raffinement de ce système conçu à haut niveau dans SPACE en une librairie de composants simulables que l'on pourrait éventuellement implémenter sur un FPGA. Elle constitue une des contributions centrales de cette recherche. Ce raffinement peut se diviser en trois parties, inspirées de la méthodologie de codesign (Figure 4.1) :

- **Raffinement des communications :** À Elix et Simtek (voir ANNEXE D), implémenter le protocole SPACE par échange de messages via des modules matériels SystemC respectant les requis de la section 4.1.
- **Génération du logiciel :** À Simtek, les modifications génériques apportées au Tor et les modifications du RTOS spécifiques à un processeur pour supporter le DirectLink. Ce code compilé sous la forme d'une librairie est directement utilisable dans un environnement de développement pour FPGA.
- **Raffinement des modules matériels :** Modifications aux modules matériels de base SPACE (SpaceBaseModule) pour supporter les nouveaux canaux de

communication DirectLink. Le résultat doit être valide pour un futur développement au niveau RTL sur FPGA.

4.3. Spécification des interfaces

Le concept du DirectLink peut s'étendre aux différents niveaux de SPACE, Elix et Simtek. À Elix, cela permettrait de relier deux modules SPACE ensembles. À Simtek, deux modules ou un processeur et un module. Ce lien doit être reproductible sur FPGA, sa spécification est d'autant plus importante.

4.3.1. Constitution du module

Le module SPACE (SpaceBaseModule) devient la base du support du DirectLink. Pour assurer la compatibilité à plusieurs niveaux d'abstraction de même qu'à différentes technologies, on y ajoutera, en plus du lien actuel pour bus partagé (SpaceModuleIF) des liens directs rapides d'entrée et de sortie (Figure 4.2) qu'on appellera SpaceInStreamDirectLinkIF (SISDL) et SpaceOutStreamDirectLink (SOSDL). Pour faciliter le développement, les deux forment une paire inséparable, dont on peut décider de brancher ou non (i.e. un port peut rester flottant). Cela permet notamment de créer un processus de calcul intense style pipeline. Ainsi, le module aura son port de contrôle (celui qui existe actuellement) ou de débit moyen, puis des ports de données à haut débit.

Tous les ports sont contrôlés par les fonctions d'accès *ModuleRead* et *ModuleWrite* et le routage se passe à l'intérieur du SpaceBaseModule.

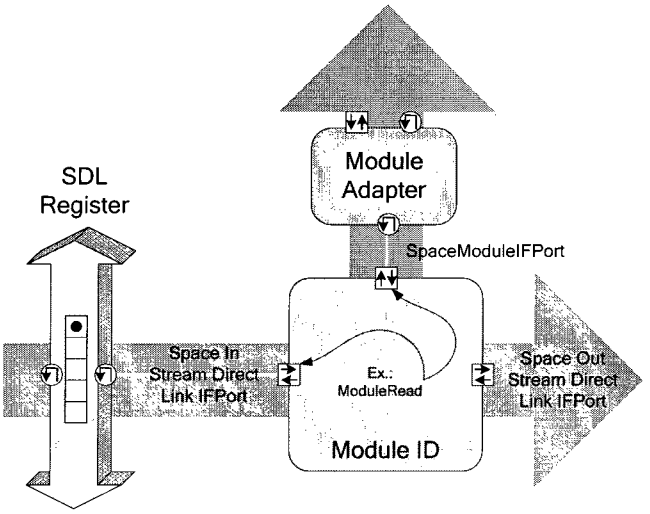


Figure 4.2 – Lien de contrôle et lien de données

4.3.2. Disponibilité des interfaces

L'interface de contrôle sera rendue accessible par un canal de communication ou un bus. Il ne pourra y avoir qu'une seule interface de ce type par module. D'abord pour assurer un seul point de configuration du module pour des applications types, puis pour des fins de compatibilité avec la structure en place (*backward compatibility*). L'interface de données à haut débit (le SISDL et le SOSDL) pourra être ajoutée autant de fois que nécessaire. Ainsi un module aura par défaut *n* interfaces (*n* étant une valeur statique connue au moment de la configuration).

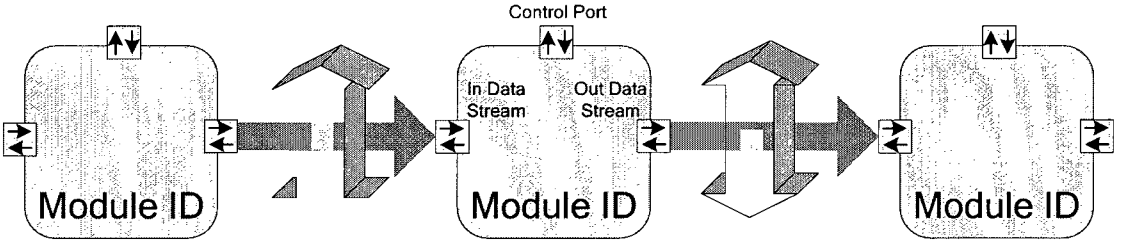


Figure 4.3 – Exemple 1 : Lien direct SDL style pipeline

Une paire est réservée pour 1 module, donc si on veut brancher deux modules en lien direct pour un traitement style pipeline (Figure 4.3), il faudra réserver deux paires. Cela facilite le design en simulation et n'influence pas la génération sur la carte (aucune allocation inutile de matériel). Il sera aussi possible d'interconnecter deux modules pour un échange bidirectionnel haut débit (Figure 4.4)

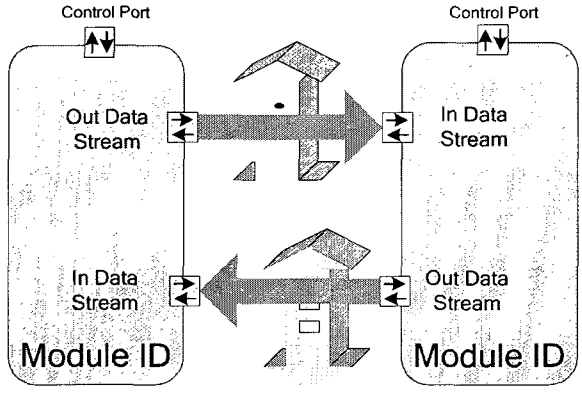


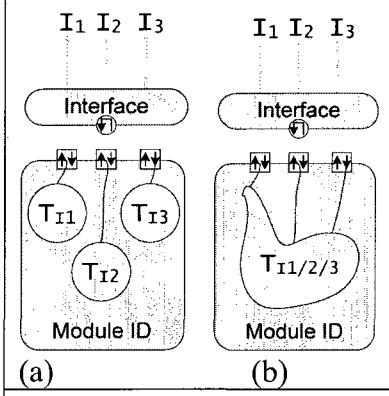
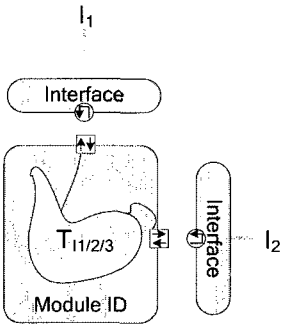
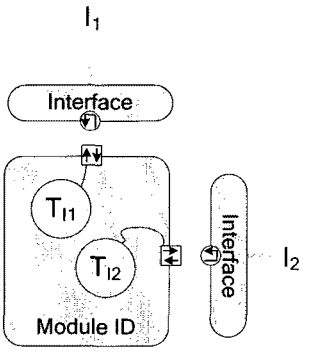
Figure 4.4 – Exemple 2 : Lien direct pour des échanges rapides inter modules

Dans le cas présent, on pourra fixer par exemple *n* à 8, soit le nombre de paires In/Out. Puisque même si en théorie on pourrait supporter autant qu'on en voudrait, il y a une limite physique relativement basse à cette possibilité, causée par la superficie du FPGA.

4.3.3. Évaluation du problème interface/contrôle

Le problème du contrôle des interfaces, i.e. comment les processus internes à un SpaceBaseModule manipulent lesdites interfaces, implique des modifications à SpaceLib. Voici un tableau qui compare les différences entre les méthodes possibles à appliquer pour créer de nouvelles interfaces dans les modules SPACE.

Tableau 4.1 – Comparaison des méthodes de contrôle des interfaces

A		B	C
Groupement monolithique des interfaces		**Séparation des interfaces / Processus de calcul unique**	**Séparation des interfaces / Processus de calcul multiples**
1 thread par port	1 thread pour tous les ports	1 thread pour tous les ports	1 thread par port
			
(a)	(b)		
+ Fonctionne avec le processus de synthèse de SpaceBaseModule pour Cynthesizer[1] + SpaceLib accepte (b) sans changement important et (a) avec quelques changements. − Complexité matérielle importante. − Réduit les performances à cause du goulot : on ne peut traiter qu'une seule interface à la fois		+ Performance accrue par rapport à A ± Complexité matérielle acceptable ± SpaceLib peut supporter cela avec des changements d'envergure moyenne − La synthèse ne peut pas supporter présentement plusieurs interfaces	+ Plus faible complexité matérielle par rapport à A et B + Performance maximale ± SpaceLib doit subir les mêmes changements que B − SpaceLib ne pourra pas supporter ce module en logiciel. − La synthèse ne peut pas supporter présentement plusieurs interfaces	

[1] Le processus de synthèse se base sur les travaux de Laurent Moss [72] du CIRCUS pour transformer automatiquement un module SPACE en module VHDL

L'approche (B) *Séparation des interfaces avec processus de calcul unique* a été retenue pour être implémentée dans SpaceLib. Elle permet une réutilisation des modules en logiciel et augmente la bande passante en matériel.

4.4. Connexions module/module HW-HW

La connexion de base du DirectLink s'établit entre deux modules matériels.Comme les liens possèdent l'interface SISDL et SOSDL, la connexion est directe et l'utilisation d'adaptateur n'est pas nécessaire. Les données sont stockées à même le canal, le SDLShiftRegister (section 4.7.2). La version matérielle pour FPGA pourrait utiliser

directement le lien FSL ou une variante personnalisée pour réduire l'utilisation des ressources.

4.5. Connexions module/module HW/SW ou SW/HW

Les communications logicielles/matérielles ou matérielles/logicielles nécessitent un ajustement du fait que chaque processeur utilise une technologie différente pour se connecter directement à un point matériel externe. De plus, un pilote (driver) est requis pour gérer les communications, notamment avec le registre d'interruptions, dans le cas où les communications bloquantes sont nécessaires. SPACE Codesign supporte deux technologies dont chacune possède ses particularités : (1) le µBlaze se connecte par FSL et (2) le PowerPC se connecte par FCB.

Pour supporter correctement ces technologies, une spécification particulière et unique à chaque processeur est requise. Le principe demeure fondamentalement le même cependant, ce qui limite les modifications à apporter.

4.5.1. Cas du µBlaze

Le µBlaze possède des ports de coprocesseurs nommés FSL et branchés directement sur son pipeline d'exécution. Des instructions spécialisées permettent d'y accéder. Le µBlaze possède 8 ports FSL maîtres pour écrire seulement et 8 ports SFL esclaves pour lire seulement. Afin de bien supporter les concepts de SPACE, on unira un port maître et un port esclave pour représenter un DirectLink. Chaque lien permettra ainsi d'unir un module logiciel et un module matériel. Dans le cas du µBlaze, cela limite à 8 le nombre de DirectLinks disponibles. Ainsi, le lien de données à haut débit se transforme en un lien coprocesseur pour les connexions logicielles/matérielles. On peut considérer cela un lien rapide de processeur, puisqu'il accélère forcément les accès de ce dernier.

Afin de pouvoir harmoniser les connexions, un adaptateur de protocole convertira l'interface du module matériel (SDL) en interface FSL, permettant ainsi au lien module/µBlaze d'exister. Comme le FSL est une FIFO qui stocke l'information, l'adaptateur n'est qu'un *wrapper* qui ne conserve aucune donnée. Dans d'autres cas, comme celui du PowerPC, le canal ne contient pas d'éléments de stockage, ainsi c'est l'adaptateur qui devrait s'en préoccuper. Afin de supporter les communications bloquantes, on reliera le gestionnaire d'interruption au µBlaze et on réservera deux interruptions par lien. Ainsi, la routine d'interruptions du RTOS pourra savoir de qui proviennent les communications.

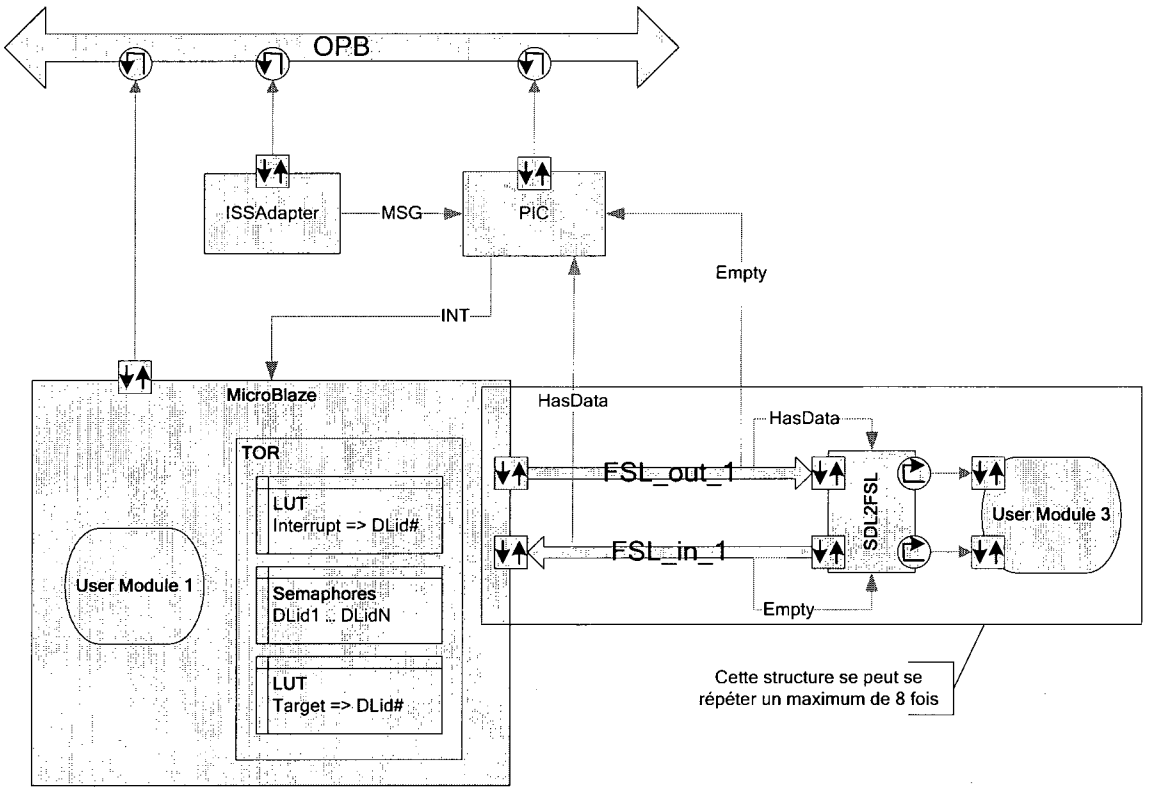


Figure 4.5 – Schématisation du raffinement DirectLink pour µBlaze

Pour garantir le bon fonctionnement du protocole SPACE, le lien doit être exclusif à deux modules et ne pas partager ce dernier :

- **Lien 1 à 1** : les données reposent dans le FIFO tant qu'on ne les lit pas, et lorsqu'on les lit, elles sont consommées par le module à ordonnancer ;
- **Lien n à 1** : même si cela réduit le nombre de liens FSL nécessaires (1 par module matériel), le destinataire pourrait ne pas être apte à réceptionner les données, causant ainsi un transfert de ces données lues du FIFO vers la mémoire. Dans ce cas, les gains du lien directs sont gâchés et l'utilisation d'une mémoire partagé est plus simple et fait tout aussi bien l'affaire.

Il est à noter que les communications bloquantes requièrent un certain mécanisme afin d'assurer leur support :

- Les signaux Empty des liens FSL de sortie du µBlaze doivent être connectés au PIC. Dans le cas d'une écriture bloquante SW-HW, le µBlaze sera donc informé par interruption de la lecture du message lorsque la FIFO du canal est vide.
- Les signaux HasData des liens FSL d'entrée du µBlaze doivent être connectés au PIC. Dans le cas d'une lecture bloquante SW-HW, le µBlaze sera donc informé par interruption de l'arrivée d'un message lorsque la FIFO du canal contient au moins une donnée.
- Le pilote du µBlaze doit contenir une table associant chaque interruption à une interface FSL. Le gestionnaire d'interruptions est donc capable de déterminer quel canal doit être traité en interrogeant le PIC lors d'une interruption.
- Un sémaphore doit être créé pour chaque lien point-à-point (i.e. paire bidirectionnelle FSL) pour assurer la synchronisation lors des communications bloquantes. Une solution alternative serait l'utilisation des fonctions Suspend et Resume du RTOS, pour désactiver/activer l'exécution des tâches.
- Le pilote du µBlaze doit contenir une table associant tous les IDs des modules HW aux liens point-à-point sur lequel ils sont connectés (i.e. numéro d'interface FSL).

Par soucis de minimiser le nombre d'interruptions, les interruptions des liens point-à-point sont masquées en permanence sauf lors de l'appel à une communication bloquante, qu'elle soit en écriture ou en lecture. De plus, étant donné l'unicité des modules aux extrémités des liens et les restrictions imposées par la nature du canal de communication, l'inclusion d'un en-tête dans les messages SPACE devient inutile. Les performances du lien sont donc accrues, en utilisant un modèle flot de données.

4.5.2. Particularités du µBlaze

Les instructions spéciales d'accès FSL ***getfsl*** ou ***putfsl*** sont bloquantes. Si le lien FSL est vide ou plein, l'exécution du µBlaze est suspendue (*pipeline stall*) empêchant ainsi tout changement de contexte entre les différentes tâches du système. Ce cas de figure est inacceptable : si une tâche bloque sur l'attente de données, les autres tâches doivent pouvoir s'exécuter. Il faut donc gérer l'état du lien de manière logicielle dans le Tor et utiliser les instructions non bloquantes ***ngetfsl*** et ***nputfsl***.

4.5.3. Cas du PowerPC405

Contrairement au µBlaze, le PowerPC405 ne possède pas de liens dédiés haut débit à proprement parler. Il possède une seule interface APU sur laquelle peut se connecter le bus FCB. Toutes les transactions passent par le bus FCB. Puisque ce bus ne peut contenir aucune donnée en tampon, des FIFOs sont présentes dans des FCB Adapter reliant les modules matériels au bus. Le lien point à point est donc virtuel.

À chaque lien virtuel, identifié par un identificateur numérique unique **DLid**, est associé deux numéros de registre APU qui seront utilisés par les instructions spécialisées prédéfinies de lecture/écriture (voir section suivante). Chaque numéro représente un canal différent du lien virtuel:

- Canal de donnée
  - Canal de transmission des données
  - Numéro de registre est formé par (DLid << 1) | 1
- Canal de contrôle
  - Permet de connaître l'état des FIFO internes à l'adaptateur
  - Permet de changer le mode d'interruption.
  - Numéro de registre est formé par (DLid << 1) | 0

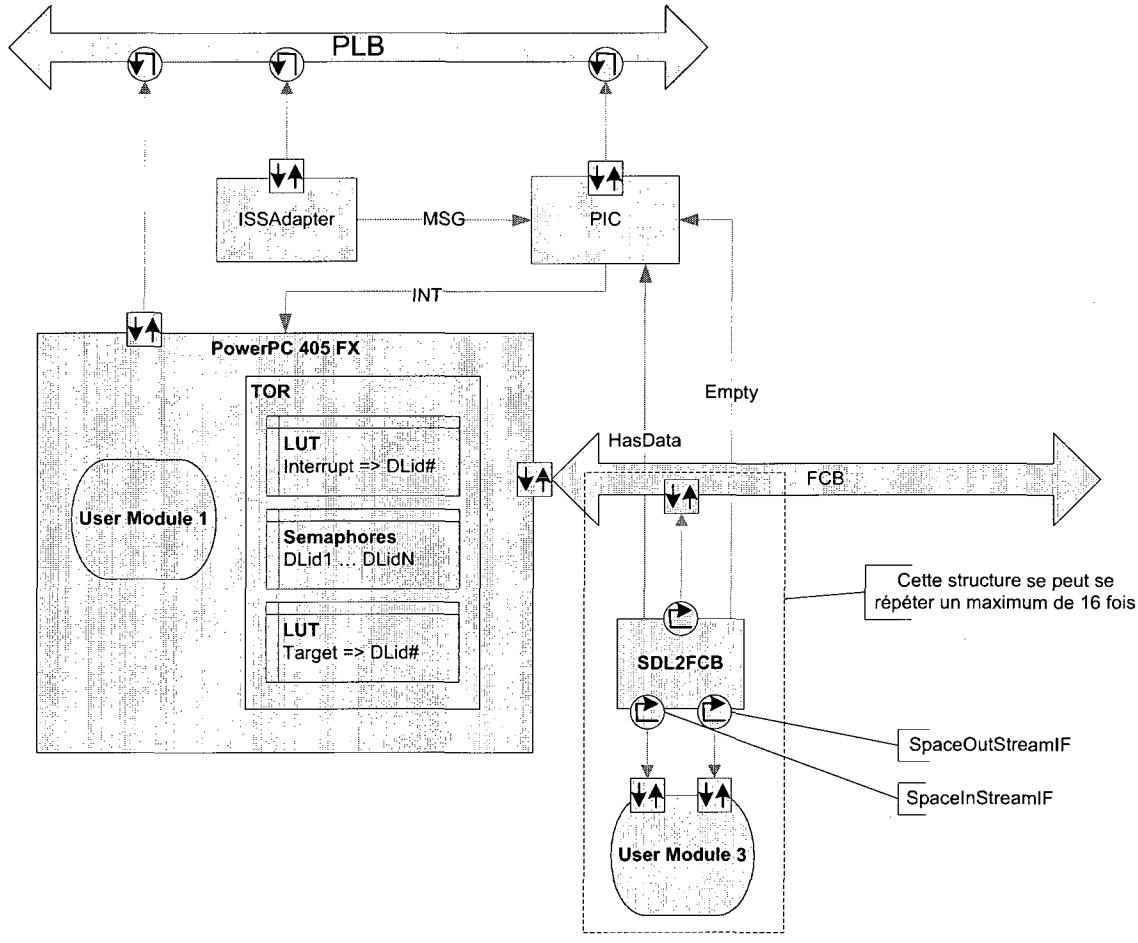


Figure 4.6 – Schématisation du raffinement DirectLink pour PowerPC405 FX

L'adaptateur peut opérer dans deux modes d'interruption :

- **EnoughData** : Une interruption sera générée dès que le FIFO a l'espace suffisant pour l'écriture d'un message en entier ou que le FIFO est vide.
- **EmptyOnly** : Une interruption sera générée seulement lorsque le FIFO est vide.

Il est à noter que les communications bloquantes requièrent un certain mécanisme afin d'assurer leur support :

- Les signaux *Empty* des adaptateurs doivent être connectés au PIC. Dans le cas d'une écriture bloquante SW→HW, le PPC sera donc informé par interruption de la lecture du message lorsque le FIFO du canal est vide ou qu'il y a un espace suffisant pour écrire un message.
- Les signaux *HasData* des adaptateurs doivent être connectés au PIC. Dans le cas d'une lecture bloquante SW←HW, le PPC sera donc informé par interruption de l'arrivée d'un message lorsque le FIFO du canal contient au moins une donnée.
- Le pilote du PPC doit contenir une table associant chaque interruption à un numéro de lien avec lequel sera formé le numéro de registre APU selon le canal désiré. Le gestionnaire d'interruption est donc capable de déterminer quel canal doit être traité en interrogeant le PIC lors d'une interruption.
- Un sémaphore doit être créé pour chaque lien point-à-point virtuel afin d'assurer la synchronisation lors des communications bloquantes. Une solution alternative serait l'utilisation des fonctions *Suspend* et *Resume*, pour désactiver/activer l'exécution des tâches.
- Le pilote du PPC doit contenir une table associant tous les IDs des modules HW aux liens point-à-point virtuels sur lequel ils sont connectés (i.e. numéro d'interface FSL).

Par soucis de performances, les interruptions des liens point-à-point sont masquées en permanence sauf lors de l'appel à une communication bloquante, qu'elle soit en écriture ou en lecture.

De plus, étant donné l'unicité des modules aux extrémités des liens et les restrictions imposées par la nature du canal de communication, l'utilisation d'un en-tête de message devient inutile. Les performances du lien sont donc accrues, en utilisant un modèle *data stream*.

4.5.4. Particularités du PowerPC405

L'APU prédéfinit plusieurs instructions Load/Store afin de transférer des données entre le système de mémoire du PowerPC (D-Cache ou mémoire DPLB/DOCM) et un coprocesseur. Chacune de ces instructions possède 5 bits pour déterminer dans quel registre du coprocesseur les données sont lues ou écrites (Tableau 4.2). Il y a donc 32 registres disponibles. En utilisant 2 registres (canal de données et de contrôle) par lien virtuel, il y a donc 16 DirectLink possible sur le PowerPC405.

Tableau 4.2 – Format des instructions prédéfinies STWFCMUX et LWFCMUX

Format PPC X – Bits																															
Opcode primaire						Registre FCM					Adresse de base					Décalage					Opcode secondaire										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Instruction STWFCMUX																															
31						FCM5					rA					rB					711										0
Instruction LWFCMUX																															
31						FCM5					rA					rB					583										0

Tous les transferts autorisés sont de 32 bits. Seulement deux instructions seront donc utilisées pour les communications point-à-point :

- **stwfcmux (*Store word with update indexed*) :** Cette instruction envoie la donnée contenue dans le registre FCM5 à l'adresse (rA + rB) & (~3) et met à jour le registre rA avec cette adresse.
- **lwfcmux (*Load word with update indexed*) :** Cette instruction envoie la donnée contenue à l'adresse (rA + rB) & (~3) dans le registre FCM5 et met à jour le registre rA avec cette adresse.

Ces deux instructions bloquent le pipeline du PowerPC405 tant que le coprocesseur ne signale pas la fin de son exécution. Aucun autre registre que rA n'est modifié lors de l'exécution. Il est donc impossible de savoir si la lecture/écriture dans le FIFO du FCB Adapter a réussi (FIFO plein, FIFO vide). C'est pourquoi un canal de contrôle est ajouté pour connaître l'état du lien et changer son mode de génération des interruptions.

4.6. Connexions module/module SW/SW

Un des avantages du DirectLink est qu'il permet la connexion virtuelle directe entre deux tâches logicielles s'exécutant sur des processeurs distincts. Cette section présente les possibilités offertes sur le Virtex4 de Xilinx.

4.6.1. Architecture homogène µBlaze

Le mécanisme logiciel/matériel proposé en 4.5.1 est valable pour connecter deux µBlaze ensemble sans aucun module supplémentaire (Figure 4.7).

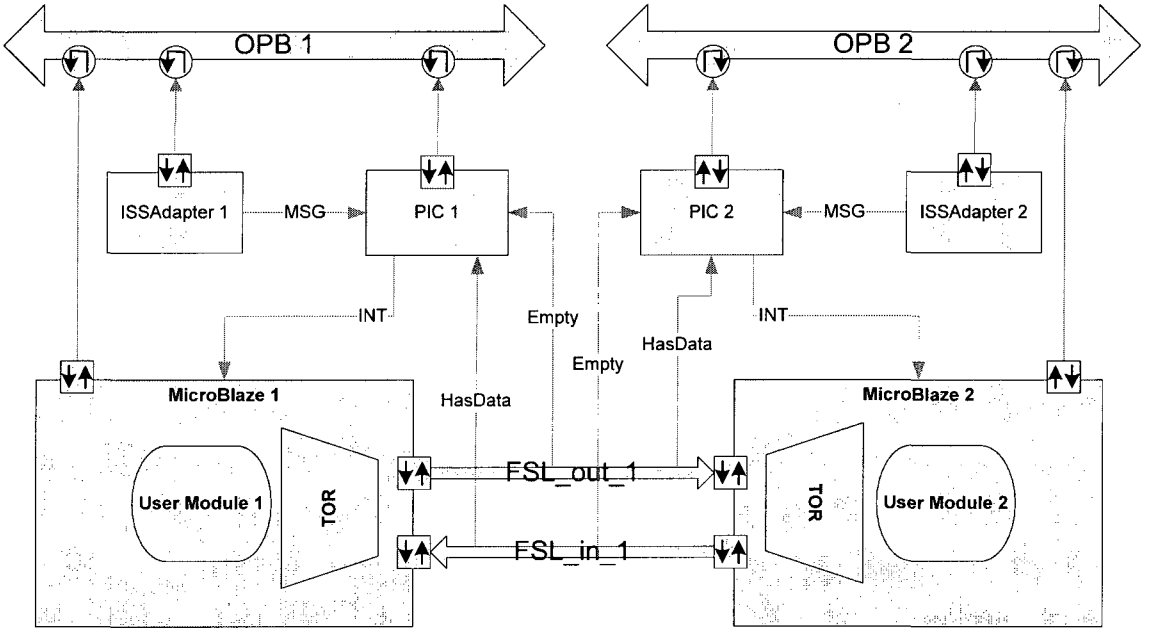


Figure 4.7 – Interconnexion de deux µBlaze par DirectLink

Aucun adaptateur de protocole n'est requis puisque les deux µBlaze utilisent le protocole FSL qui stocke les données directement dans le lien. Aucune modification au Tor, sauf celle déjà apportées en 4.5.1, n'est nécessaire.

4.6.2. Architecture homogène PowerPC405

Il est aussi possible d'interconnecter deux PowerPC405 via un nouveau composant *FCB-FCB Channel Bridge* (Figure 4.8)

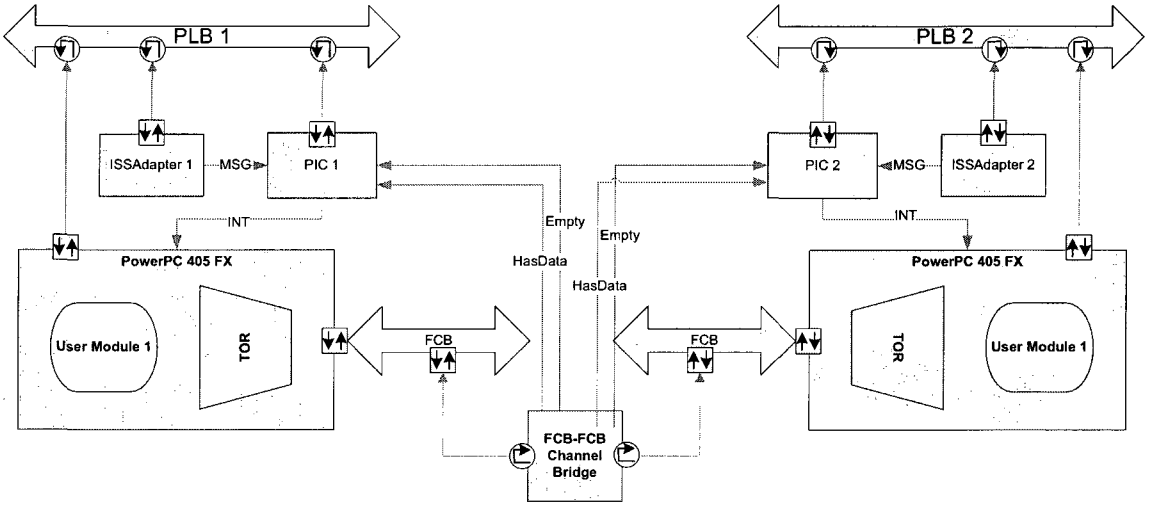


Figure 4.8 – Interconnexion de deux PowerPC405 par DirectLink

Aucun adaptateur de protocole n'est requis puisque les deux PowerPC utilisent le mécanisme APU. Cependant, un pont est nécessaire pour stocker les messages SPACE et générer les interruptions pour les communications bloquantes. Aucune modification, sauf celle déjà apportées en 4.5.3, au Tor n'est nécessaire.

4.6.3. Architecture hétérogène µBlaze-PowerPC405

Le support de systèmes multiprocesseurs hétérogènes PowerPC405-µBlaze est aussi assuré via le composant *FCB-FSL Channel Bridge* (Figure 4.9). L'utilisation d'un lien FSL est ici exclue. Le PowerPC nécessite certains signaux que le FSL ne génère pas et afin d'éviter d'avoir des tampons de données à 2 endroits différents (Brigde et lien FSL), il faut connecter le composant directement aux interfaces FSL du µBlaze. Ce pont est aussi nécessaire pour stocker les messages SPACE et générer les interruptions pour les communications bloquantes. Aucune modification, sauf celle déjà apportées aux Tor en 4.5.1 et 4.5.3, n'est nécessaire.

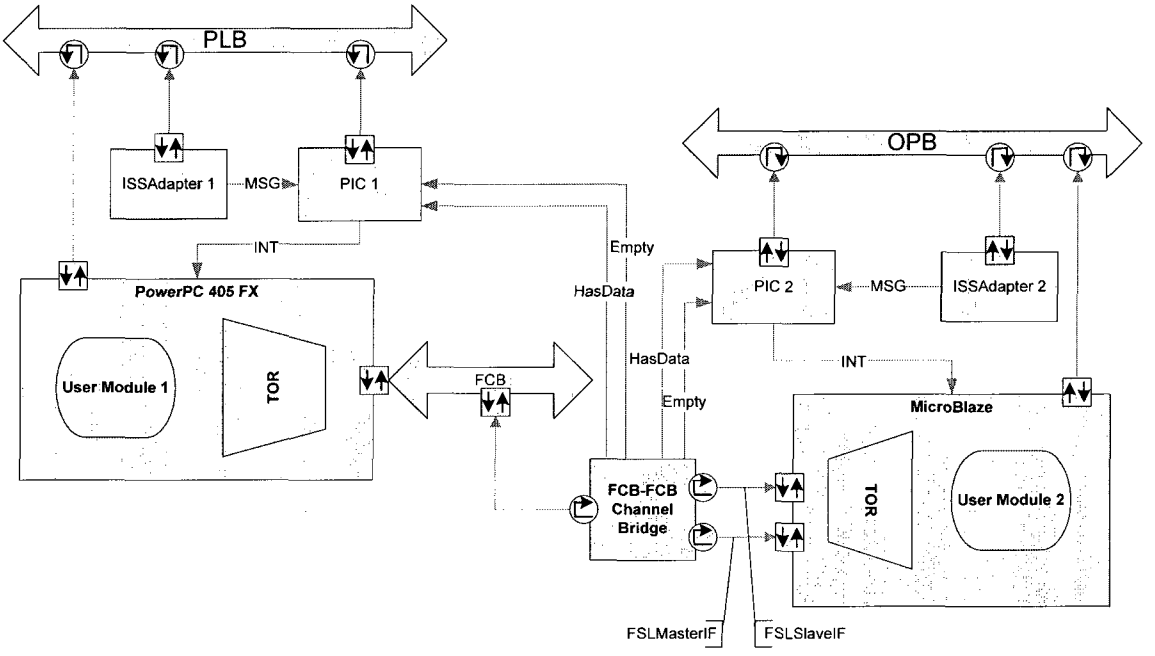


Figure 4.9 – Interconnexion d'un µBlaze et d'un PPC405 par DirectLink

4.7. Design des composants SpaceLib

Une part importante du travail de recherche est le développement de modules SystemC s'intégrant à SpaceLib et mettant en place le nouveau protocole de communication du DirectLink.

4.7.1. Interfaces

Afin d'implémenter le DirectLink, deux interfaces matérielles de base doivent être définies pour supporter le modèle décrit à la section 4.1. Ces interfaces sont des classes C++ abstraite héritant de l'interface SystemC.

Ces deux interfaces (*SpaceInStreamIF* et *SpaceOutStreamIF*), dont le code se trouve ci-dessous, modélisent l'aspect flot de données du DirectLink en fournissant deux méthodes distinctes pour les communications bloquantes et non bloquantes. Elles

prennent en paramètres un pointeur vers le tampon de données contenant le message SPACE et la taille de ce tampon.

Code de SpaceInStreamIF

```
class SpaceInStreamIF :
public  sc_interface
{
public:
    virtual eStatus nReadStream (
        void* vpData32,
        long ulDataLength8
        ) = 0;

    virtual eStatus ReadStream (
        void* vpData32,
        long ulDataLength8
        ) = 0;
};
```

Code de SpaceOutStreamIF

```
class SpaceOutStreamIF :
public  sc_interface
{
public:
    virtual eStatus nWriteStream (
        void* vpData32,
        long ulDataLength8
        ) = 0;

    virtual eStatus WriteStream (
        void* vpData32,
        long ulDataLength8
        ) = 0;
};
```

Figure 4.10 – Code des interfaces In/Out Stream

4.7.2. SDLShiftRegister

Le SDLShiftRegister (Figure 4.11) est le composant matériel de base pour la communication point à point entre deux modules fonctionnels ou matériels. Il permet cette connexion sans l'utilisation d'adaptateurs, implémentant les interfaces *SpaceInStreamIF* et *SpaceOutStreamIF*. Il contient une FIFO synchrone ou asynchrone (pour relier deux domaines d'horloge) afin de stocker les messages SPACE.

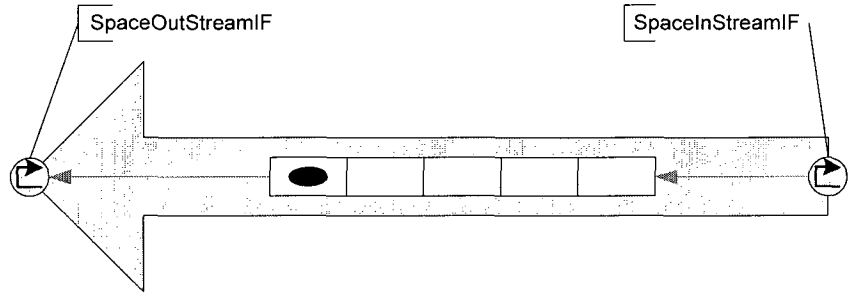


Figure 4.11 – Architecture du SDLShiftRegister

Il offre une latence configurable selon Elix ou Simtek, il est unidirectionnel et la taille de son FIFO est configurable. Le SDLShiftRegister peut être facilement implémenté sur FPGA en utilisant la BRAM disponible dans le Virtex4. Sa sémantique est similaire à celle du lien FSL, mais élimine certains signaux inutiles. Les communications bloquantes sont supportées, de la même façon que pour le ModuleAdapter (i.e. le thread du module peut bloquer lors d'appels de primitives de communication sur des objets de synchronisation SystemC).

4.7.3. SDLToFSLAdapter

Le SDLToFSLAdapter est le composant matériel de base pour la communication point à point entre un module matériel et un processeur µBlaze.

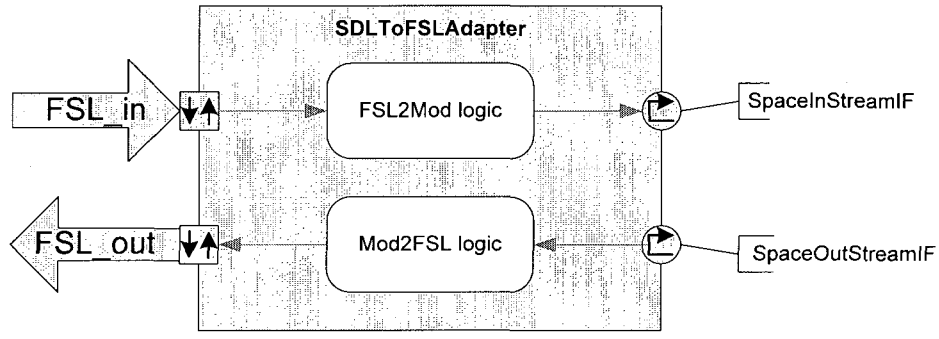


Figure 4.12 – Architecture du composant SDLToFSLAdapter

Il opère une conversion de protocole quasi directe entre les requêtes SpaceStream et FSL et est utilisable de manière unidirectionnelle ou bidirectionnelle, en connectant une seule des interfaces ou les deux. Cet adaptateur sérialise les messages SPACE de longueur *n* provenant des interfaces *stream* en effectuant *n* accès sur les interfaces FSL.

4.7.4. SDLToFSBAdapter

Le SDLToFSLAdapter est le composant matériel de base pour la communication point à point entre un module matériel et un processeur PowerPC 405 FX via le bus de communication FCB. Il stocke les messages SPACE dans deux FIFO internes de tailles configurables. Il opère une conversion complexe entre l'analyse d'instructions prédéfinies de l'APU et le protocole SpaceStream. Il est chargé avec l'identificateur du DirectLink qu'il implémente.

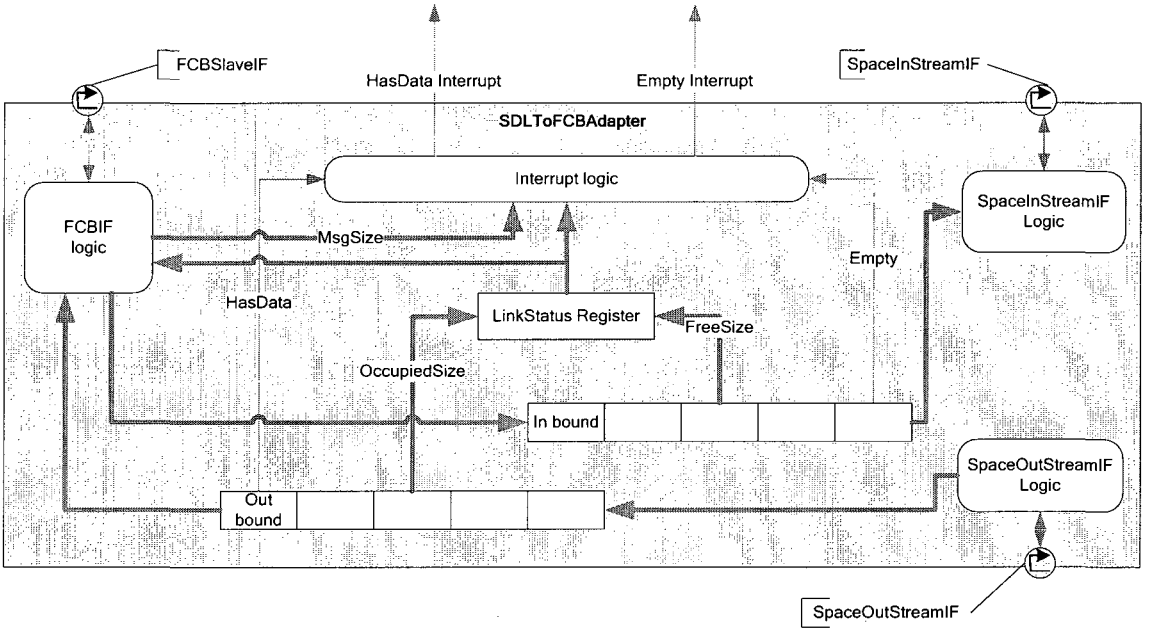


Figure 4.13 – Architecture du composant SDLToFCBAdapter

Il offre les caractéristiques suivantes :

- Les FIFO doivent générer les signaux suivants :
  - **FreeSize** : espace libre dans le FIFO ;
  - **OccupiedSize** : espace occupée dans le FIFO ;
  - **Empty** et **HasData** : FIFO vide et FIFO non vide respectivement.
- LinkStatus Register est la concaténation de OccupiedSize et FreeSize.

- LinkStatus = (OccupiedSize << 16) | FreeSize
- La valeur de LinkStatus peut être lue via le canal de contrôle.

- Le mode d'opération de InterruptLogic peut être changé via le canal de contrôle

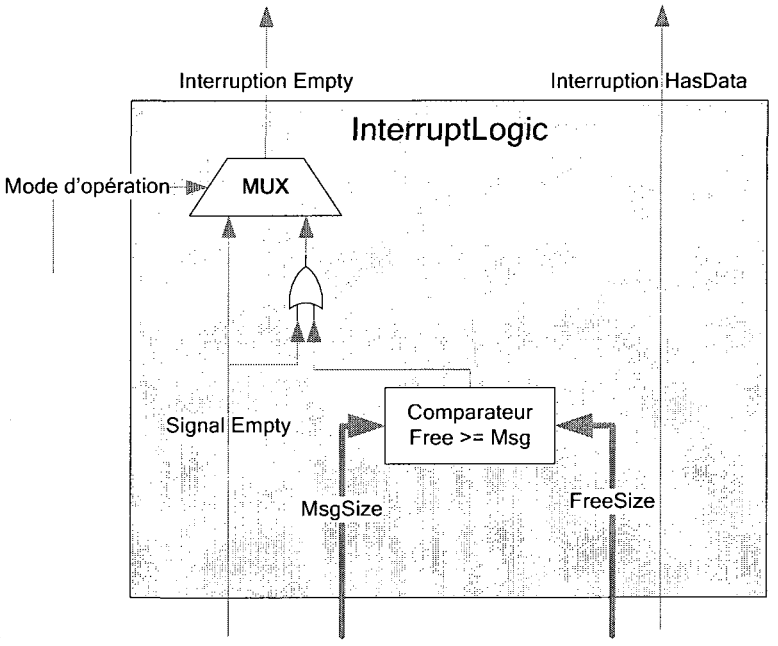


Figure 4.14 – Logique d'interruption pour le composant SDLToFCB

La logique d'interruption illustrée à la Figure 4.14 s'occupe de contrôler la génération de l'interruption *Empty* selon le mode d'opération. Le registre *MessageSize* peut être chargé via le canal de contrôle et représente la taille du message à écrire. En mode EnoughData, l'interruption *Empty* est déclenchée lorsque *FreeSize* est supérieur ou égal à *MessageSize.*

Dans la Figure 4.13, le bloc *FCBIF Logic* s'occupe d'analyser les instructions qui passent sur le bus FCB et déterminer s'il doit les exécuter. Il obéit à l'arbre de décision illustré à la Figure 4.15.

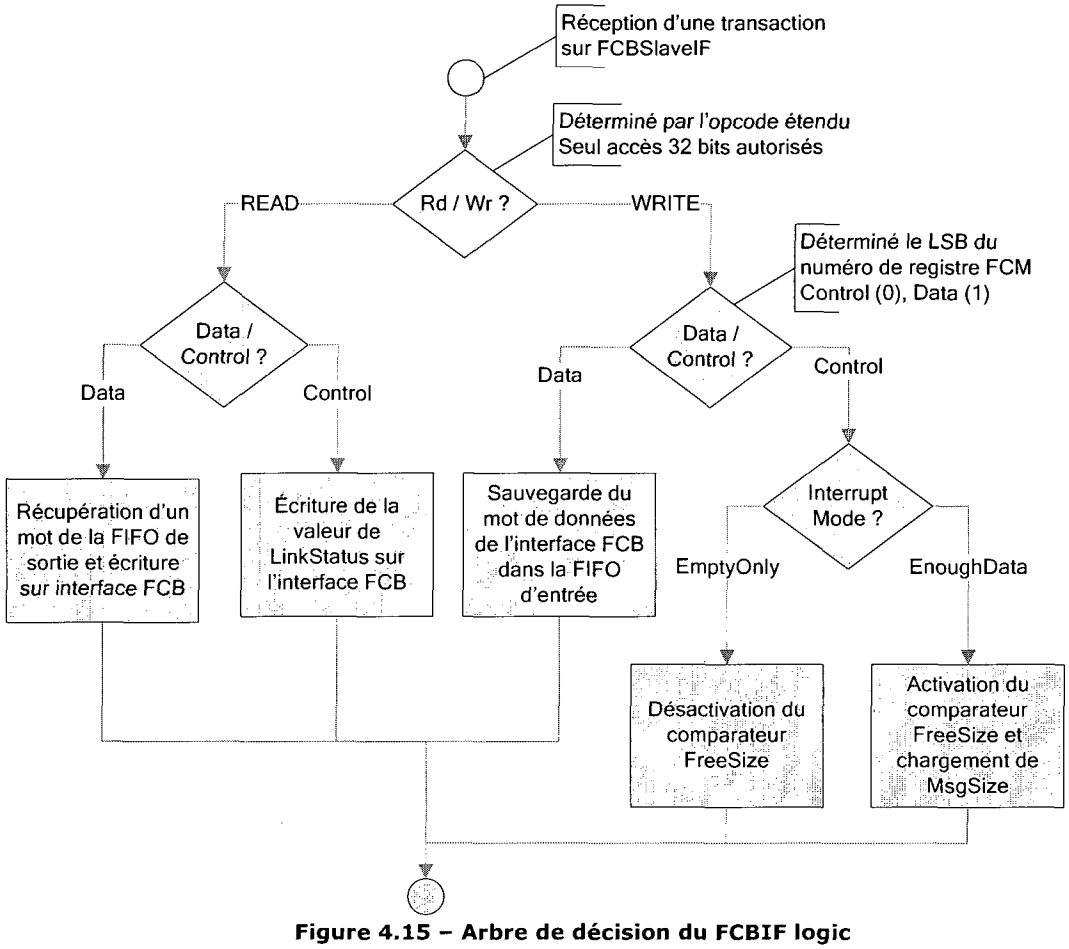


Figure 4.15 – Arbre de décision du FCBIF logic

Une lecture du canal de contrôle fourni toujours la valeur du registre *LinkStatus*. Lors d'une écriture au canal de contrôle, deux commandes peuvent être chargées :

- WAKEUP_ENOUGHDATA_MODE dont le mot de code est ((1 << 31) | (MsgSize & 65535))
- WAKEUP_EMPTYONLY_MODE dont le mot de code est 0.

4.7.5. FCBFCBChannelBridge

Le *FCB-FCBChannelBridge* permet l'établissement d'un lien point à point entre deux PowerPC405 FX via le bus de communication FCB. Il stocke les messages SPACE dans deux FIFO internes de tailles configurables. Il est chargé avec les deux identificateurs des DirectLink qu'il implémente sur chaque PowerPC405 FX.

Il reprend l'architecture du SDLToFCBAdapter (Figure 4.13) et remplace la logique des interfaces SpaceStream par un dédoublement de la logique du FCBIF.

4.7.6. FCBFSLChannelBridge

Le *FCB-FSLChannelBridge* permet l'établissement d'un lien point à point entre un PowerPC405 FX et un µBlaze. Il stocke les messages SPACE dans deux FIFO internes de tailles configurables. Il est chargé avec l'identificateur du DirectLink qu'il implémente sur le PowerPC405 FX.

Il reprend l'architecture du SDLToFCBAdapter (Figure 4.13) et remplace la logique des interfaces SpaceStream par de la logique pour le protocole FSL.

4.7.7. SpaceBaseModule

Le SpaceBaseModule de SpaceLib est modifié pour refléter le modèle de la section 4.3 :

- Ports SystemC supplémentaires pour supporter l'interface point à point via les interfaces SpaceInStreamIF et SpaceOutStreamIF. Les fonctionnalités point-à-point deviennent standard à tout nouveau module créé.
- Ajout de traitement supplémentaire pour décider quelle interface de communication prendre, le lien point à point ou le bus de contrôle partagé. Ce

traitement concerne les primitives de communication *ModuleRead()* et *ModuleWrite()*.

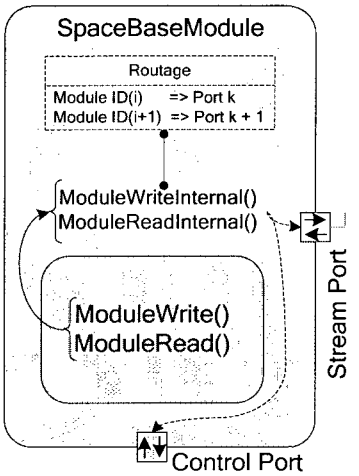


Figure 4.16 – SpaceBaseModule amélioré

Deux tables de routage sont encapsulées dans SpaceBaseModule servant à prendre la bonne interface, point à point ou bus, pour les données en entrée (via ModuleRead) ou en sortie (via ModuleWrite). À chaque identificateur de module possible i ($0 < i < 255$) est associée l'interface de contrôle ou une des n interfaces DirectLink.

Trois méthodes sont disponibles à l'interne :

- *InitRoutingTable* : Initialisation d'une table de routage, par défaut toutes les communications passent par l'interface de contrôle ;
- *AddRoute* : Ajout d'une entrée à la table de routage ;
- *DirectLinkWhichChannel* : Détermine l'identificateur de l'interface à prendre pour communiquer avec un module donné (via son identificateur de module).

Les tables de routage sont construites explicitement lors de l'instanciation de la simulation SystemC et non pas automatiquement lors de la phase d'élaboration de la simulation. Cette dernière méthode aurait été coûteuse au plan de la complexité de SpaceLib.

Deux méthodes sont disponibles via l'interface du SpaceBaseModule pour attacher un composant SystemC implémentant les interfaces *stream*. Ces méthodes déterminent une interface libre, attachent le composant et ajoutent l'entrée correspondante à la table de routage du module. Ces méthodes peuvent être appelées manuellement par le concepteur mais sont aussi gérées par le modèle haut niveau DirectLink présenté à la section.

4.8. Implications au niveau de la pile logicielle

Au niveau logiciel, un module utilisateur SPACE n'a plus de ports SystemC puisque celui-ci est encapsulé en tant que tâche logicielle. Donc la seule implication est d'analyser le message et de savoir si, dans le cas d'un message pour un module HW, celui-ci est envoyé vers l'interface point à point ou l'interface de contrôle (typiquement le bus).

La Figure 4.17 illustre l'architecture du Tor avant les modifications pour le support du DirectLink. Il comprend un composant *Gestionnaire des communications*, aussi appelé SWBus (*Software Bus*), qui route les messages entre le logiciel et le matériel. Si un message SPACE est destiné à module logiciel contenu sur le processeur, le message est stocké dans une FIFO logicielle associé au module. Dans le cas contraire, le message est écrit sur le bus à l'adresse du module matériel destinataire.

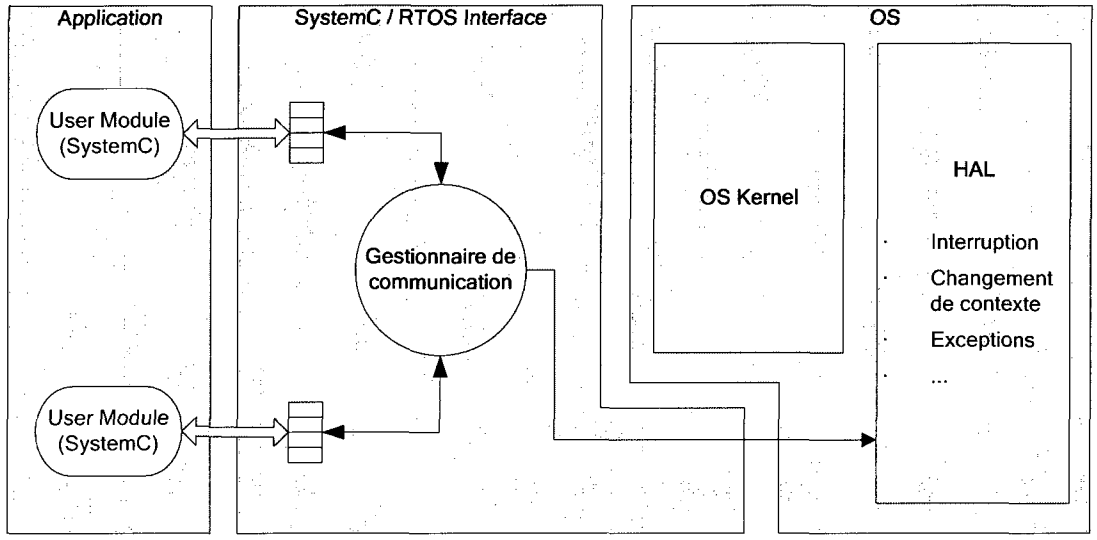


Figure 4.17 – Architecture logicielle SPACE pré-DirectLink

La Figure 4.18 illustre l'architecture du Tor après les modifications apportées pour supporter le DirectLink.

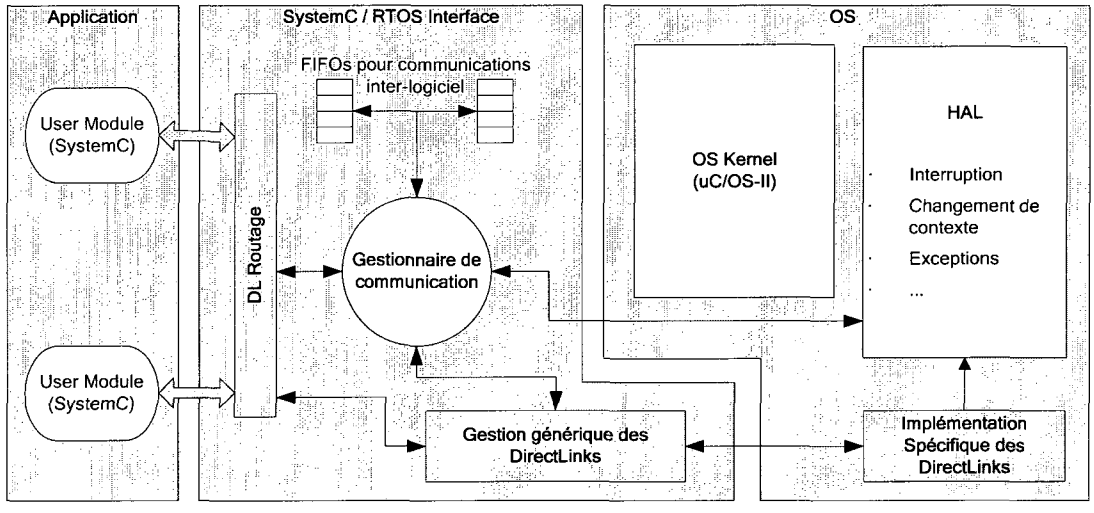


Figure 4.18 – Architecture logicielle SPACE post-DirectLink

Lors d'appels aux primitives de communication ModuleRead ou ModuleWrite, une étape de routage des messages SPACE doit s'effectuer. Une table de routage similaire à celle contenue dans le *SpaceBaseModule* est donc construite. Elle associe à chaque identificateur *i* de module possible (0 < i < 255) une structure `DirectLinkCoprocInfos` contenant les informations nécessaires à la gestion d'un DirectLink :

- L'identificateur de canal du DirectLink (registre FCM, numéro de lien FSL, etc.) ;
- Le numéro de l'interruption Empty pour ce DirectLink ;
- Le numéro de l'interruption HasData pour ce DirectLink.

La table est construite via l'API logiciel du DirectLink avec la fonction *RegisterDirectLink*. Cette fonction est appelée pour chaque DirectLink lors de l'initialisation du système logiciel dans la fonction *DirectLinkInitializationHook*. Cette fonction est stockée dans un fichier C++ généré par SpaceStudio.

Deux fonctions de gestion générique des DirectLink sont intégrées sur SWBus, soit *InternalStreamRead* et *InternalStreamWrite*. Lorsqu'il est déterminé qu'un message doit transiter via un DirectLink, ces fonctions sont appelées. Elles transfèrent le message à la fonction d'implémentation spécifique propre à chaque processeur en spécifiant le canal DirectLink à utiliser et gèrent le caractère bloquant des communications par le masquage-démasquage des interruptions et la manipulation des sémaphores de blocage de tâches.

Les fonctions d'implémentation spécifiques sérialisent les messages SPACE selon le protocole propre au processeur (sections 4.8.1 et 4.8.2). C'est ici que les caractéristiques telles les instructions spécialisées ou les canaux de communication haut débit sont utilisées pour transférer le message. Afin d'éviter la spéculation de données, i.e. annulation d'une instruction de transfert après la phase d'exécution dans le pipeline

résultant en une donnée stockée dans une FIFO du DirectLink, les interruptions sont désactivées pendant tout le transfert du message.

4.8.1. Protocoles µBlaze

La sérialisation et l'envoi des messages SPACE via DirectLink doit suivre certains protocoles propres à chaque processeur afin d'assurer leur intégrité, leur transfert à bas niveau et leur mode bloquant. Quatre cas de figures sont envisageables pour le logiciel sur µBlaze.

4.8.1.1.Lectures bloquantes et non-bloquantes

Pour une lecture non-bloquante, si aucune donnée n'est contenue dans le lien FSL, la lecture échoue. Dans le cas contraire, la quantité requise de données est lue de manière non-bloquante (ngetfsl). Les interruptions du lien FSL sont masquées pendant toute l'exécution de la routine.

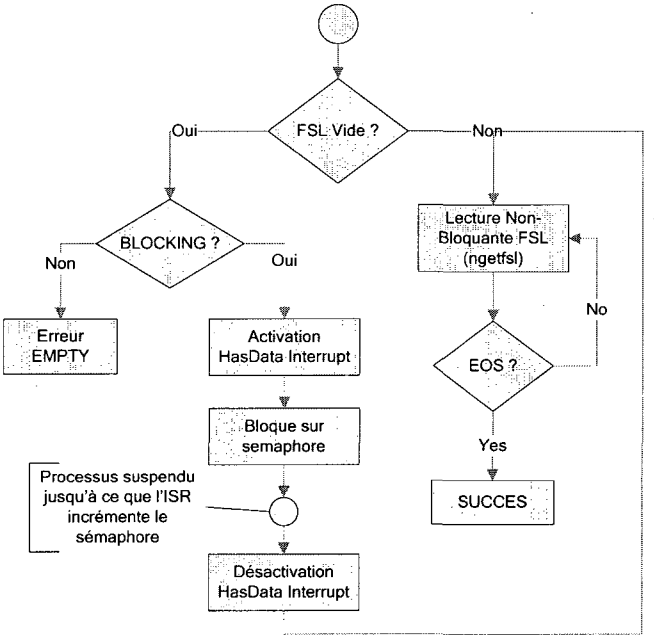


Figure 4.19 – Diagramme de décision pour les lectures via FSL

Dans le cas d'une communication bloquante, si aucune donnée n'est présente dans le lien FSL au moment de l'appel, l'interruption HasData du lien est activée et le processus bloque sur un sémaphore de synchronisation. Il sera réveillé par l'ISR du µBlaze dès qu'un message commence à être stocké dans le lien. Dans le cas où un message – ou une portion de message en cours de transfert – est présent, la quantité requise de données est lue de manière non-bloquante (ngetfsl).

4.8.1.2.Écritures bloquantes et non-bloquante

Pour une écriture, les données sont écrites de manière continue et non-bloquante (nputfsl). Dès que le FIFO est plein, le processus bloque dans l'attente qu'elle se vide complètement, puis reprend le transfert jusqu'à ce que le message soit transmis en entier. Le processus ne bloquera pas une deuxième fois, car puisque le FIFO s'est vidé, le message présentement transmis est en cours de lecture par le module à l'autre extrémité du lien. Si le FIFO est plein dès la première écriture, il serait aussi possible de quitter la routine en retournant un code d'erreur FIFO_FULL.

Le paradoxe d'une écriture dite non-bloquante qui bloque lorsque le canal est plein est inévitable avec l'utilisation d'un lien FSL, car :

- Il est impossible de connaître l'espace libre dans le lien FSL ;
- Il est impossible de savoir quand le lien se libérera lorsqu'il est plein ;
- Lorsque le FIFO est plein, il est inacceptable d'effectuer un *polling* du lien car les autres tâches doivent pouvoir s'exécuter ;
- Il est inacceptable de quitter la routine dès que le FIFO est plein, car le message ne serait pas transmis en entier. Ceci occasionnerait un crash du système.

Bien sûr, le fait de bloquer sur un sémaphore et d'exécuter un ISR ralentit considérablement les performances du système. Cependant, la taille des FIFOs est

ajustable en simulation. Une taille minimale sera choisie afin d'éviter que ce cas de figure se produise.

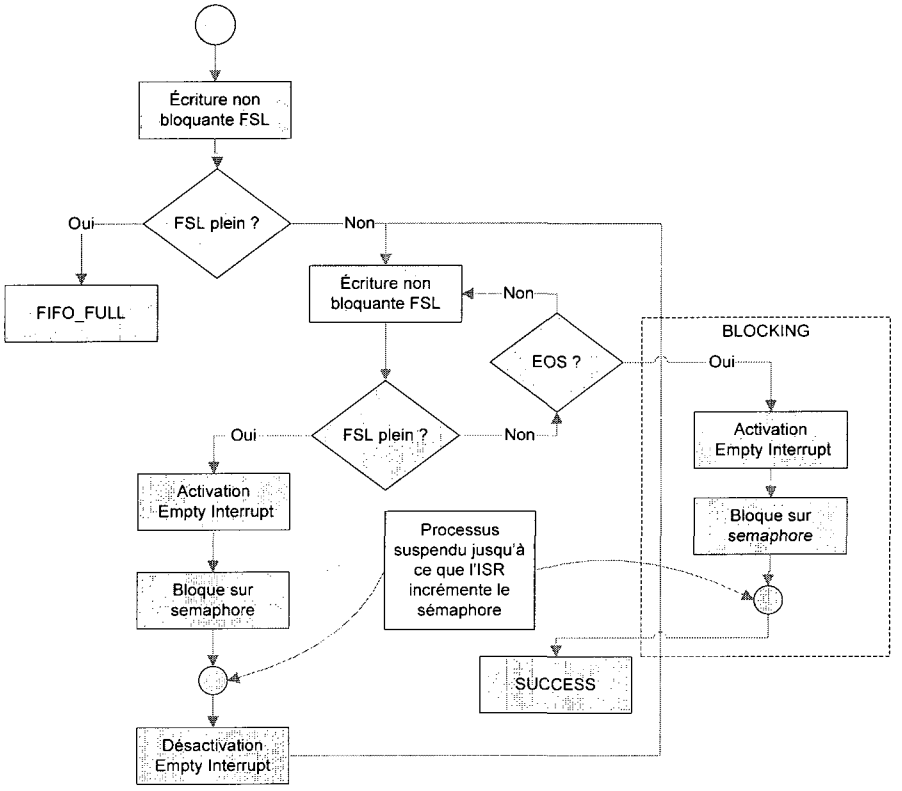


Figure 4.20 – Diagramme de décision pour les écritures via FSL

Dans le cas bloquant, le concept est le même que l'écriture non-bloquante. Une étape supplémentaire est ajoutée à la fin de la transmission du message afin d'attendre que le FIFO se vide complètement, indiquant que le message transmis a été lu correctement.

4.8.2. Protocoles PowerPC405

Comme pour le µBlaze, la sérialisation et l'envoi des messages SPACE via DirectLink doit suivre certains protocoles. Quatre cas de figures sont envisageables pour le logiciel sur PowerPC405 FX.

4.8.2.1.Lectures bloquantes et non-bloquantes

Pour toutes lectures, le registre d'état du lien doit être lu via le canal de contrôle pour vérifier la présence de données dans le lien. Pour la lecture non-bloquante, si aucune donnée n'est contenue dans le lien, la lecture échoue. Dans le cas contraire, la quantité requise de données est lue via le canal de données. Les interruptions du lien sont masquées pendant toute l'exécution de la routine.

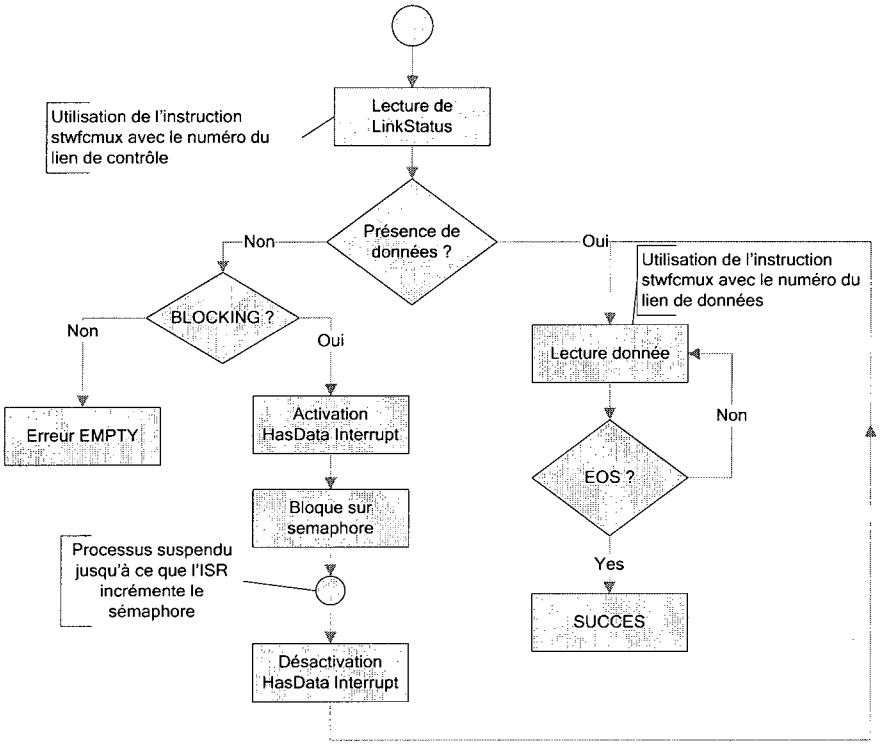


Figure 4.21 – Diagramme de décision pour les lectures via FCB

Dans le cas d'une communication bloquante, si aucune donnée n'est présente dans le lien au moment de l'appel, l'interruption HasData du lien est activée et le processus bloque sur un sémaphore de synchronisation. Il sera réveillé par l'ISR externe du PowerPC dès qu'un message commence à être stocké dans le lien. Dans le cas où un

message – ou une portion de message en cours de transfert – est présent, la quantité requise de données est lue via le canal de données.

4.8.2.2.Écritures bloquantes et non-bloquantes

Toutes les écritures commencent par une lecture du registre d'état du lien. Puisque l'écriture peut bloquer le pipeline du processeur, il faut s'assurer que le transfert ne sera pas interrompu. Une vérification est donc effectuée à savoir s'il y a assez d'espace libre dans la FIFO du canal. Si c'est le cas, le message est transmis via le canal de données. Dans le cas contraire, une commande WakeUp (section 4.7.4) est envoyée au DirectLink pour que l'interruption EMPTY soit activée lorsque le lien est vide ou qu'il contient assez d'espace pour stocker le message en entier. Les interruptions sont réactivées sur le PowerPC et le processus bloque sur un sémaphore de synchronisation. Le message sera transféré lors du réveil de la tâche.

Dans le cas d'une écriture bloquante et suite à la complétion de la transmission, une commande WakeUp est envoyée du DirectLink pour que l'interruption EMPTY soit activée lorsque le lien est vide. Les interruptions sont réactivées sur le PowerPC et le processus bloque sur un sémaphore de synchronisation.

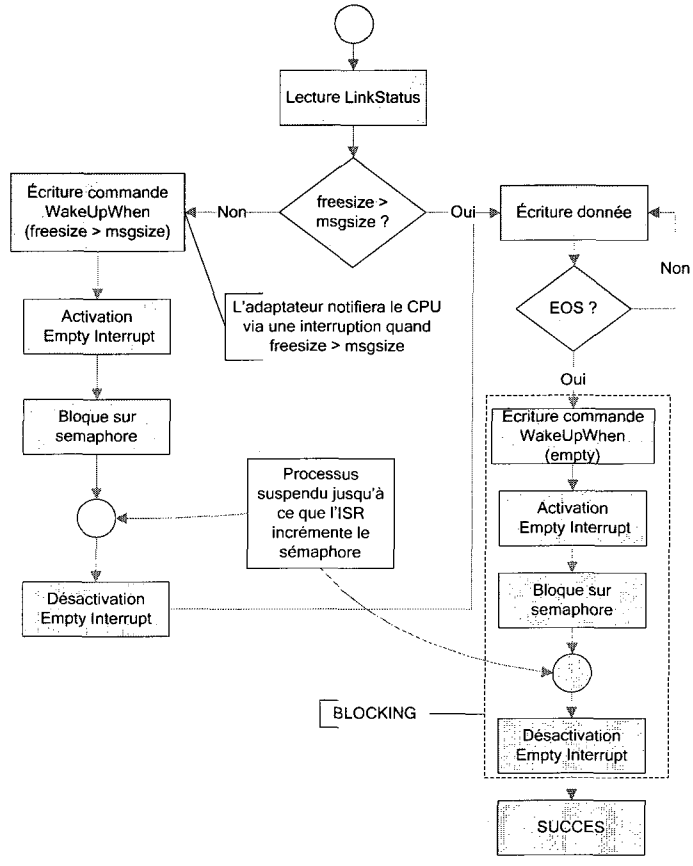


Figure 4.22 – Diagramme de décision pour les écritures via FCB

4.9. Abstraction du DirectLink dans SPACE

Une des étapes de conception du DirectLink était l'établissement d'un modèle à haut-niveau masquant la complexité des implémentations possibles telles que discutées dans la section 4.1.

4.9.1. Modélisation

Pour faciliter la conception des architectures de communication de MPSoC dans SPACE, un modèle abstrait du DirectLink doit être mis à la disposition du designer. Cette abstraction (Figure 4.23) est à la fois un modèle de visualisation du DirectLink qui peut

être utilisé dans SpaceStudio, un outil de conceptualisation du mécanisme et une classe C++ (module SPACE de SpaceLib) pouvant être instanciée dans une simulation SPACE.

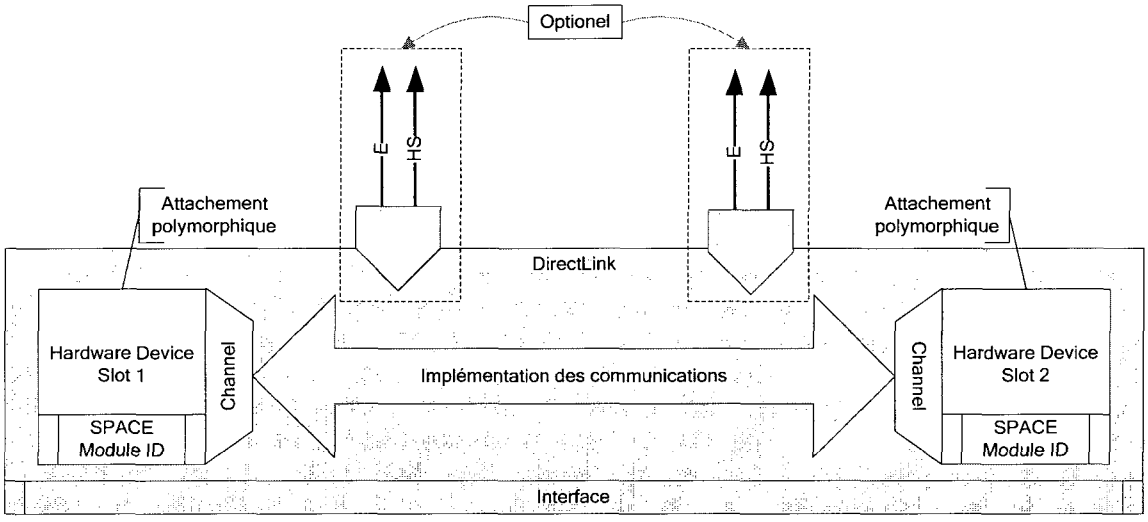


Figure 4.23 – Schématisation de l'abstraction DirectLink

Le DirectLink peut se modéliser comme une boîte noire masquant une implémentation d'un mécanisme de communication point-à-point (souvent une conversion de protocole entre des mécanismes de communication haut-débit).

Il accepte la connexion de **deux** périphériques matériels (*HardwareDeviceSlot*, i.e. PowerPC405, µBlaze, SpaceBaseModule) sans se soucier du type de la connexion. Un mécanisme de sélection de l'implémentation basé sur une librairie s'effectuera en fonction du type des deux périphériques attachés, si une telle implémentation existe. À chaque périphérique sont associées deux valeurs :

- *SpaceModuleID* : Identificateur numérique du module SPACE attaché. Dans le cas d'un microprocesseur, il s'agit de l'*ID* du module logiciel s'exécutant sur ce microprocesseur et auquel le DirectLink est associé. Il sert à identifier les signaux d'interruption propres à chaque périphérique.

- *ChannelID* : Identificateur numérique du canal sélectionné sur le périphérique matériel pour établir la connexion du DirectLink. Chaque DirectLink doit être connecté sur un canal différent. Dans les cas présentés dans ce chapitre, le canal peut se raffiner sous trois formes :
    - Numéro de port FSL dans le cas du μBlaze
    - Numéro de registre FCM dans le cas du PowerPC405 FX
    - Numéro de port *stream* dans le cas du SpaceBaseModule

Optionnellement, pour chaque périphérique, si celui-ci est un microprocesseur, le DirectLink peut fournir une paire de signaux d'interruption *Empty* et *HasData* pouvant se connecter au contrôleur d'interruption du microprocesseur. Le concepteur est libre de les connecter sur les ports de son choix.

4.9.2. Implémentation dans SpaceLib

Ce modèle est implémenté (Figure 4.24) dans SpaceLib et peut être instancié dans une simulation SPACE. La classe **DirectLink** est la façade que le concepteur utilise. Trois méthodes sont disponibles :

- *AttachDevice* : C'est une méthode polymorphique permettant d'attacher un périphérique matériel au DirectLink. Il en existe une version pour chaque type de périphérique disponible (présentement 3). À chaque méthode est associé un identificateur de type unique.
- *HasDataInterrupt* et *EmptyInterrupt* qui permettent de récupérer les signaux d'interruption optionnels associés aux identificateurs de module SPACE.

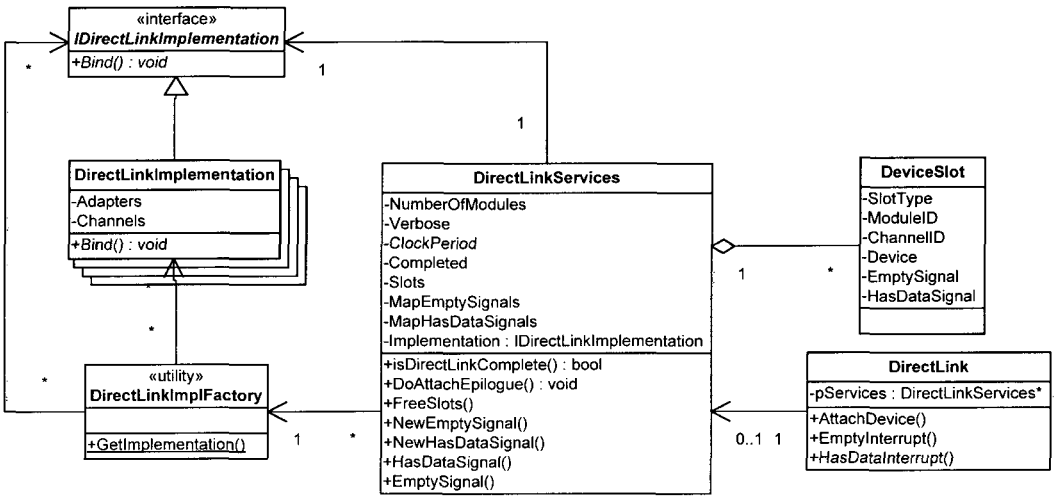


Figure 4.24 – Diagramme de classe simplifié du DirectLink

La classe **DirectLinkServices** implémente la gestion des périphériques attachés (par deux structures **DeviceSlot**), la génération des signaux d'interruption (sc_signal) et garde une référence vers l'implémentation du mécanisme de communication via l'interface **IDirectLinkImplementation**. Chaque DeviceSlot comprend l'information sur le type de périphérique, l'identificateur de module, l'identificateur de canal ainsi que des références aux objets de périphériques et de signaux.

Lorsque le lien est complet, i.e. deux périphériques sont attachés correctement, DirectLinkServices fait appel à la classe **DirectLinkImplFactory** pour sélectionner et instancier l'implémentation nécessaire dans une banque de classes d'implémentation. Chaque implémentation possède un identificateur généré de manière unique à partir des identificateurs des deux types de périphériques qu'elle peut relier. La sélection s'opère sur cet ID généré comme suit : ((1 << ComponentType1) + (1 << ComponentType2)). Une fois l'implémentation sélectionnée, la méthode **bind()** est appelée. Tous les modules SPACE nécessaires pour effectuer la conversion de protocole entre les deux périphériques matériels sont instanciés et les canaux/signaux sont connectés aux ports appropriés.

CHAPITRE 5

ANALYSE, PERFORMANCES ET DISCUSSION

Ce chapitre présente les résultats liés au fonctionnement et à l'implémentation du DirectLink pour le Virtex-4 de Xilinx. Il discute aussi de la validation empirique et qualitative du paradigme ainsi que son extensibilité à d'autres architectures.

Les latences et débits de communications du matériel au matériel, du matériel au logiciel et du logiciel au logiciel sont présentés pour les deux variantes du DirectLink, i.e. celle du PowerPC405 FX qui utilise des instructions spécialisées et celle du µBlaze utilisant des canaux de communications dédiés. Ces résultats sont aussi comparés aux méthodes conventionnelles de communication dans SPACE. Brièvement, le DirectLink permet une accélération tant du point du vue matériel que logiciel. En logiciel, il permet une réduction des latences de 54% pour l'écriture et de 96% pour les lectures. L'interface matérielle SDL (*stream*) permet une augmentation de la bande passante d'un facteur 3 dans le pire des cas.

La méthodologie actuelle de communication SPACE est aussi analysée et certaines améliorations sont suggérées pour améliorer la complexité et les performances globales de l'architecture de communication sous-jacente à SPACE.

5.1. Validation du paradigme DirectLink

Puisque la majorité des objectifs du DirectLink ne sont pas quantifiables, leur évaluation et leur validation passent par une approche qualitative[1]. Le DirectLink présente une

[1] Basée sur les discussions avec Cédric Migliorini suite à son utilisation du DirectLink pour accélérer une application JPEG dans le contexte de son mémoire [68]

transparence et une facilité d'utilisation élevée du point de vue matériel. L'établissement d'un lien point-à-point se complique cependant lorsqu'un module logiciel est présent. La gestion de la connectivité n'est pas uniforme entre les partitions logiciel/matériel et nécessite l'appel à la fonction *RegisterDirectLink*, ce qui viole la prémisse d'interface unique pour les deux partitions. Cette problématique est impossible à régler et est causée par le fonctionnement même de SpaceLib. Puisque dans SPACE les deux partitions sont compilées séparément, il est impossible que les propriétés du DirectLink instancié en simulation (déclaration de la partition matérielle) soient connues des logiciels applicatifs s'exécutant sur les microprocesseurs. En bref, la table de routage logicielle ne peut être construite par l'instanciation du modèle C++ du DirectLink dans une simulation. Deux solutions sont possibles pour éviter la gestion de cette complexité par l'utilisateur :

1. Inclusion du modèle DirectLink dans SpaceStudio comme paradigme de visualisation des liens point-à-point. SpaceStudio devient alors légataire du choix de l'implémentation et peut générer les fichiers des partitions matérielle et logicielle en conséquence.
2. S'inspirer des travaux du professeur Jerraya (section 2.3.3) et traiter les mécanismes de communication comme des services offerts aux logiciels applicatifs. Cette méthode fait disparaître la gestion mutuellement exclusive des deux partitions mais nécessite une refonte en profondeur de SpaceLib.

Le besoin de généricité du modèle est respecté. Il est utilisable sans modifications tant sur le PowerPC que sur le µBlaze. Le concepteur peut relier deux modules sans connaître l'existence du FSL ou de l'APU. Tel que démontré à la section 5.6, le modèle est même extensible à deux autres plateformes. L'inconvénient du modèle est son incapacité à modéliser de manière uniforme la gestion des communications bloquantes nécessitant des signaux d'interruption. Il est à noter que ces signaux sont primordiaux pour respecter les modes de communication SPACE sans engendrer des délais d'attente

active au niveau logiciel. Ils doivent cependant être connectés manuellement au périphérique de gestion des interruptions. Quatre solutions sont envisageables :

1. L'ajout d'un troisième (ou quatrième) *DeviceSlot* optionnel au modèle du DirectLink qui contiendrait le(s) contrôleur(s) d'interruption pour les microprocesseurs attachés. Le DirectLink pourrait ainsi effectuer lui-même les connexions des signaux d'interruption. Ceci requiert que tous les contrôleurs d'interruptions possèdent la même interface SystemC permettant de connecter des signaux par l'appel d'une méthode virtuelle. Dans le cas du NIOS-2 [69], c'est le processeur lui-même qui devient contrôleur et qui devrait implémenter cette interface. Ceci pourrait confondre l'utilisateur qui doit connecter le NIOS-2 au DirectLink à deux reprises pour deux fonctions distinctes.
2. Inclusion du modèle dans SpaceStudio. Cette solution déplace le problème à un niveau supérieur. De plus, l'utilisateur ne voulant pas faire appel au mode graphique se retrouve avec la même problématique. La tendance actuelle dans SPACE prône cette méthodologie. Les travaux de Nicolas Laug[2] [70] pourraient éventuellement automatiser la connexion des signaux sans créer des exceptions pour chaque implémentation possible de DirectLink.
3. Lier logiquement un microprocesseur à son contrôleur d'interruption et être capable de connecter directement un signal d'interruption via la référence au microprocesseur. Ainsi, µBlaze, PowerPC et NIOS-2 peuvent être manipulés de la même manière et il n'est pas nécessaire de référencer directement les PIC.
4. Éliminer les communications bloquantes pour le logiciel. Cette méthode peut sembler être un déni du problème, mais si l'on accepte l'hypothèse que le DirectLink doit être un lien à haut débit dans un modèle de type flot de données, l'utilisation de modes de communication bloquants SPACE n'a plus son sens. Ces communications bloquants servent plutôt à effectuer des poignées de main

[2] Étudiant du CIRCUS travaillant sur la génération automatique des interconnexions entre modules, basée sur des descriptions XML des interfaces compatibles SPIRIT.

(*handshakes*) ou de la synchronisation entre processeurs et modules et pourraient bien passer par le canal de contrôle.

L'introduction du DirectLink dans SPACE pose aussi la problématique du partitionnement à 3 choix : logiciel, matériel partagé, matériel dédié (via lien point-à-point). Quand est-il valable de transférer un module SPACE sur DirectLink ? Quand est-ce que l'augmentation de bande passante vaut la réduction de la réutilisation d'une architecture MPSoC ? Ce questionnement pourrait être abordé dans le cadre d'une extension des travaux de Laurent Moss [71] sur le partitionnement automatique basé sur des métriques de performances et des estimations de surfaces/puissance.

Le requis de diminution de la consommation de puissance est théoriquement atteint par l'utilisation de FIFOs asynchrones dans le DirectLink pour relier deux domaines d'horloges. Ce mécanisme rejoint les travaux de [40] dans le domaine des GALS (voir section 2.2.3). Le lien FSL supporte déjà la connexion à deux horloges différentes et l'APU du PowerPC possède un mécanisme de resynchronisation des signaux pour interfacer la logique FPGA ayant une fréquence inférieure à celle du cœur du processeur. Il est malheureusement impossible de valider expérimentalement cette hypothèse puisque SPACE ne supporte pas à ce jour les horloges multiples ni l'estimation de puissance d'une architecture.

5.2. Technique d'analyse des performances

La technique d'analyse pour estimer l'impact du DirectLink sur les performances d'une architecture se base sur la simulation. Toutes les données analysées sont tirées de bancs d'essais SPACE s'effectuant à un niveau hybride TF – BCA (voir ANNEXE B) à Simtek. Aucune implémentation sur FPGA Xilinx n'a été réalisée puisque les composants de la section 4.7 n'ont pas été réalisés à ce jour en VHDL.

Ces simulations ont permis d'évaluer les différentes latences intervenant lors d'appels aux primitives de communication ModuleRead et ModuleWrite bloquantes et non bloquantes. Ces latences sont mesurées pour le système conventionnel de communication dans SPACE, soit les bus partagés OPB ou PLB, et pour le DirectLink. L'objectif est de constater quel est le gain de performances atteignable par l'utilisation du DirectLink. Ces latences sont indépendantes de l'application modélisée dans SPACE dans le cas DirectLink. Dans le cas conventionnel, elles sont liées au trafic sur le bus et les méthodes d'arbitrage. Afin d'obtenir le meilleur des cas possible, les bancs de tests ne sont que l'envoi d'un module à un autre de messages dont la taille augmente progressivement. Aucun autre module n'est présent afin d'éviter le phénomène de contention.

Les temps mesurés correspondent aux temps physiques et non aux temps de simulation. Le temps physique est le temps écoulé à l'intérieur de la simulation basé sur la fréquence d'horloge du système modélisé. Les temps sont convertis en cycles d'exécution à partir de la fréquence d'horloge afin de pouvoir comparer efficacement le µBlaze et le PowerPC. Le code est instrumenté à l'aide de la fonction **sc_simulation_time()**[3] qui retourne la valeur du temps physique actuel de la simulation dans l'unité de temps sélectionnée. Par défaut, SPACE utilise la nanoseconde (SC_NS).

5.3. Performances du DirectLink

Deux types de délais interviennent lors de communications dans SPACE : (1) les latences matérielles induites par les adaptateurs de communication et les protocoles utilisés et (2) les latences logicielles induites par les mécanismes de contrôle et de gestion des communications sur les microprocesseurs.

---

[3] L'instrumentation n'affecte pas les performances à l'exécution

5.3.1. Latences matérielles

Les latences matérielles sont simples à déterminer dans le cas du DirectLink puisqu'elles sont constantes. Aucun mécanisme d'arbitrage ou phénomène de contention ne peut modifier ces valeurs. Les résultats pour la méthode conventionnelle se basent les travaux de [67].

5.3.1.1.Méthode conventionnelle par bus partagé

Dans la méthode conventionnelle, les communications passent par les adaptateurs de modules connectés au bus partagé. Ces adaptateurs sont très complexes. Au total, 13 latences différentes peuvent intervenir au niveau de ces adaptateurs [67]. Par rapport à l'utilisation directe de l'interface du bus OPB, les adaptateurs ajoutent deux cycles de latence. De plus, les communications bloquantes requièrent un transfert supplémentaire pour le message ACK.

Dans le meilleur des cas pour l'OPB, si un maître demande et obtient le bus dans le même cycle et que l'esclave affiche une latence de 1 cycle d'horloge, il est possible d'achever une écriture/lecture en 2 cycles [55]. Bien sûr, le trafic sur le bus fait en sorte que le maître ne peut pas nécessairement obtenir le bus dès qu'il le demande. Une fois l'en-tête de message transféré, il est possible de sauver les cycles d'arbitration et de demande du bus à l'aide du mécanisme de *bus lock*. Selon [67], voici les délais en cycles de transmission typiques des adaptateurs développés pour SPACE :

- Écriture non bloquante HW vers HW : $3N + 16 + \xi + D$;
- Lecture non bloquante HW de HW : $N + 6 + X$.

Où N représente le nombre de données du message, D le nombre de cycles d'attente pendant que le bus traite les requêtes des maîtres les plus prioritaires, ξ le nombre de

cycles d'attente quand l'interface esclave est occupée à traiter une requête de l'arbitre du bus OPB et X le nombre de cycles d'attente d'envoi de l'acquittement quand le message lu provient d'une écriture bloquante

Si le transfert s'effectue du matériel vers le logiciel, l'ISSAdapter intervient. Le transfert s'effectue en 2 phases :

- Envoi des données de l'adaptateur de module vers l'ISSAdapter. Ce cas revient à celui d'une écriture non bloquante HW vers HW.
- Signalement par interruption au processeur (2 cycles). Le reste du processus est logiciel et analysée dans la section suivante.

5.3.1.2.DirectLink : SDL Interface

Les communications entre deux modules SPACE de la partition matérielle via les interfaces *stream* sont les plus rapides. Elles passent par le SDLShiftRegister qui est capable d'accepter une donnée à chaque cycle en lecture (par une méthode de *prefetch*) et en écriture. Il faut donc un minimum de deux cycles pour qu'une donnée transite d'un module vers un autre.

5.3.1.3.DirectLink : FSL Interface

Le lien FSL utilisé lors de communication entre deux modules SPACE peut interfacer directement un µBlaze ou un adaptateur comme le FSL2FCB ou FSL2SDL. Selon sa spécification [57], les interfaces FSL du µBlaze prennent 2 cycles d'horloge pour effectuer un transfert d'un registre processeur à la FIFO du FSL. Idéalement, i.e. si une instruction FSL est envoyée dans le pipeline à chaque cycle, la bande passante est divisée par deux par rapport au DirectLink HW-HW (*SDLShiftRegister*).

5.3.1.4.DirectLink : APU-FCB Interface

Le lien APU-FCB, utilisé lors de communication entre deux modules SPACE, peut interfacer directement un PowerPC ou un adaptateur comme le FSL2FCB, FCB2SDL ou FCB2FCB. Selon les spécifications de l'APU pour les instructions prédéfinies de chargement/écriture, le temps d'exécution est déterminé par le temps que prend la logique du FCM à répondre à la requête.

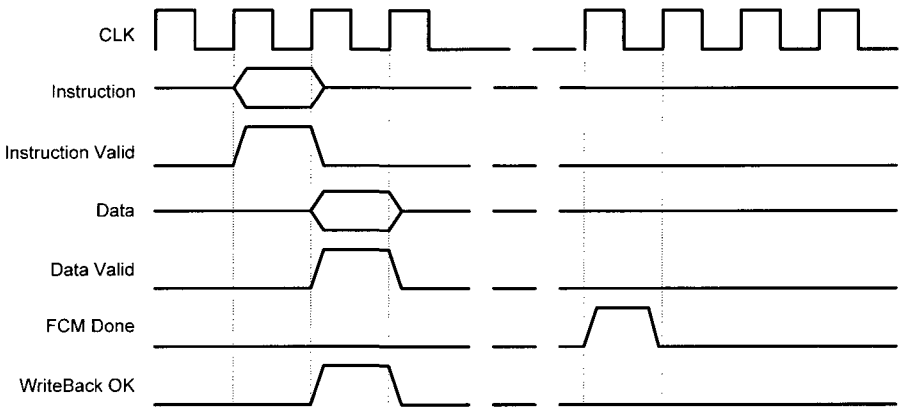


Figure 5.1 – Chronogramme d'une instruction FCM Load 32 bits

La Figure 5.1 illustre qu'une instruction d'écriture 32 bits vers un FCM (*load*) prend minimalement 3 cycles d'horloge (en supposant que le FCM fonctionne à la même fréquence que le PowerPC405 FX). C'est cette latence qui a été implémentée en simulation, puisque aucun calcul n'est nécessaire. La logique FCM ne fait que stocker la donnée dans une FIFO. De façon optimale, i.e. si une instruction *lwfcmux* est envoyée dans le pipeline à chaque cycle, la bande passante est divisée par trois par rapport au DirectLink HW-HW (*SDLShiftRegister*).

L'APU supporte aussi le transfert de *doublewords* (64 bits) et de *quadwords* (128 bits). Avec ces modes, il est possible de transférer à chaque cycle les mots de 32 bits constituant le *double* ou le *quad* après le préambule requis de 1 cycle. Le transfert d'un

quadword double la bande passante. Ces modes ne sont cependant pas implémentés en simulation.

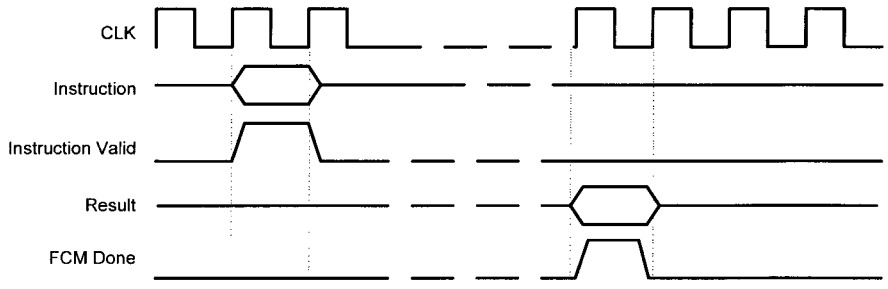


Figure 5.2 – Chronogramme d'une instruction FCM Store 32 bits

La Figure 5.2 illustre qu'une instruction de lecture 32 bits d'un FCM (*store*) prend minimalement 2 cycles d'horloge (en supposant que le FCM fonctionne à la même fréquence que le PowerPC405 FX). C'est cette latence qui a été implémentée en simulation, puisque aucun calcul n'est nécessaire. La logique FCM ne fait que lire la donnée d'une FIFO. De façon optimale, i.e. si une instruction *stwfcmux* est envoyée dans le pipeline à chaque cycle, la bande passante est divisée par deux par rapport au DirectLink HW-HW (*SDLShiftRegister*).

5.3.1.5.Comparaisons

Pour les communications matériel/matériel, l'interface SDL (*stream*) du SpaceBaseModule permet la transmission d'un mot de 32 bits à tous les cycles d'horloge. En reprenant les équations de la section 5.3.1.1, avec des délais d'attente nuls, ceci permet une augmentation significative de la bande passante (Tableau 5.1) par rapport à la méthode conventionnelle. Plus les messages sont petits, plus le taux d'utilisation du bus partagé diminue et plus il est avantageux d'utiliser un DirectLink, surtout lors d'écritures vers d'autres modules.

Tableau 5.1 – Augmentation de la bande passante utile selon l'interface DirectLink et l'opération effectuée

	Taille du message	
Gain en débit	**N = 1**	**N →∞**
Lecture SDL	7.00	1.00
Écriture SDL	19.00	3.00
Lecture FSL	3.50	0.50
Écriture FSL	9.50	1.50
Lecture APU	2.33	0.33
Écriture APU	6.33	1.00

On remarque que pour des tailles de message élevées, l'utilisation du bus permet un plus grand débit de données que les interfaces matérielles FSL et APU du DirectLink. Cependant, toutes ces interfaces sont contrôlées de manière logicielle – contrairement au SDL qui est contrôlé par un processus matériel – et c'est à ce niveau que le gain de performance est atteint.

5.3.2. Latences logicielles

Au niveau logiciel, trois latences différentes entrent en ligne de compte : (1) le temps de transmission d'un mot de 32 bits vers l'interface matérielle, (2) le temps de préparation du message et de contrôle du lien et (3) le temps d'exécution des interruptions.

Les interruptions interviennent dans les cas suivants :

- Méthode conventionnelle :
  - Réception d'un message par le logiciel (pour les lectures bloquantes ou non bloquantes). Le temps d'exécution est variable selon la taille du message à recevoir et à stocker dans une FIFO logicielle.
  - Réception d'un acquittement par le logiciel (pour les écritures bloquantes). Le temps d'exécution est constant.
    - µBlaze : Le temps d'exécution est évalué à 1905 cycles
    - PowerPC405 : Le temps d'exécution est évalué à 1441 cycles

- Méthode DirectLink :
    - Déblocage d'une tâche (pour les lecture et écritures bloquantes). Le temps d'exécution est constant.
        - µBlaze : Le temps d'exécution est évalué à 2988 cycles.
        - PowerPC405 : Le temps d'exécution est évalué à 2241 cycles.

Cette analyse a été effectuée pour les deux méthodes et sur les deux processeurs embarqués disponibles dans SPACE.

5.3.2.1.Délais d'écriture

Pour évaluer les latences logicielles en écriture pour le µBlaze et le PowerPC405, les délais d'envoi pour des messages dont la taille croît de 1 à 25 mots de 32 bits a été mesurée. La profondeur des FIFO matérielles est de 30 mots de 32 bits. On remarque que le comportement est linéaire (ANNEXE G). Trois cas ont été étudiés :

- **Conventionnel** : Méthode sans DirectLink. Les messages sont sérialisés puis envoyés sur le bus partagé. Le bus est en mode *lock* pendant tout le transfert.
- **DirectLink** : Méthode utilisant le DirectLink. Les messages sont sérialisés et envoyés sur l'interface matérielle (APU ou FSL).
- **DirectLink Optimal** : Méthode utilisant le DirectLink avec des instructions (APU prédéfinies ou FSL) pour lesquelles les opérandes sont fixes.

Une problématique ayant apparue lors des simulations est que les instructions **putfsl** et **getfsl** du µBlaze V6.0 prennent plus de cycles qu'une transmission par le bus. Ceci est causé par le fait que le format des instructions n'accepte qu'un littéral comme numéro de lien FSL (e.g. getfsl 5, r31). Or, le numéro du lien est déterminé dynamiquement à l'exécution. Il faut donc utiliser un tableau de pointeurs de fonction, l'index du tableau représentant le numéro d'un lien. Chaque fonction implémente une instruction FSL avec

un numéro de lien différent. Cette méthode est cependant très lente comparativement à un simple accès bus.

DirectLink Optimal représente la performance qu'aurait le DirectLink si chaque appel de fonction était remplacé par une simple instruction FSL à opérande fixe par une méthode d'analyse/substitution de code avant la compilation du logiciel. Une autre façon d'obtenir le même résultat est l'utilisation du µBlaze version 7.0 qui supporte des instructions FSL à opérandes dynamiques (numéro du lien stocké dans un registre).

Tableau 5.2 – Temps d'exécution pour écriture sur µBlaze

Type	Préparation (cycles)	Transmission (cycles)	Interruption (cycles)
DirectLink	1998	362	2988
DirectLink Optimal	1965	145	2988
Conventionnel	4325	164	1905

On remarque que le temps de préparation des messages diminue de 54% avec l'utilisation du DirectLink. Ceci est du au fait qu'aucun en-tête de message ne doit être construit et transmis et qu'une multitude de *sanity checks* (i.e. vérification de la taille du message, de la validité de la destination, etc.) sont évités. Dans le cas des temps de transmission, la méthode optimale DirectLink par instructions dynamiques permet une réduction de 11.5% du temps, ce qui peut sembler minime. Il est à noter que la comparaison s'effectue avec un bus sans contention utilisé dans le meilleur des cas. La méthode DirectLink souffrant du problème d'opérandes statiques multiplie par 2.2 le temps de transmission d'un mot de 32 bits. Cependant, la transmission reste plus rapide que la méthode conventionnelle jusqu'à une taille de message de 12 mots grâce au gain significatif obtenu au niveau de la préparation du message. Une taille si grande est peu utilisée et il en résulte que le DirectLink est plus performant en écriture dans la grande

majorité des cas[4]. En fait, les gains les plus importants s'effectuent au niveau des lectures.

Pour le PowerPC, la même problématique est apparue lors des simulations. Les instructions prédéfinies APU prennent plus de cycles qu'une transmission par le bus. Ceci est causé par le fait que le format des instructions n'accepte qu'un littéral comme numéro de registre FCM (e.g. lwxfcm 5, r31). Or, le numéro du lien est déterminé dynamiquement à l'exécution. Il faut donc utiliser un tableau de pointeurs de fonction, l'index du tableau représentant le numéro d'un lien. Chaque fonction implémente une instruction prédéfinie APU avec un numéro de lien différent. Cette méthode est cependant très lente comparativement à un simple accès bus.

DirectLink Optimal représente la performance qu'aurait le DirectLink si chaque appel de fonction était remplacé par une simple instruction prédéfinie APU à opérande fixe par une méthode d'analyse/substitution de code avant la compilation du logiciel. Une autre façon d'obtenir le même résultat est la création et l'utilisation d'instructions UDI totalement personnalisées qui supporteraient des opérandes dynamiques (numéro du lien stocké dans un registre).

Tableau 5.3 – Temps d'exécution pour écriture sur PowerPC405

Type	Préparation (cycles)	Transmission (cycles)	Interruption (cycles)
DirectLink	818	191	2241
DirectLink Optimal	858	52	2241
Conventionnel	1021	90	1441

On remarque que le temps de préparation des messages diminue de 20% avec l'utilisation du DirectLink. Ceci est du au fait qu'aucun en-tête de message ne doit être

[4] Discussions avec Maxime De Nanclas et Luc Filion du groupe de recherche CIRCUS et œuvrant sur le développement de SPACE.

construit et transmis et qu'une multitude de *sanity checks* sont évités. Dans le cas des temps de transmission, la méthode optimale DirectLink par instructions dynamiques permet une réduction de 42% du temps. La méthode DirectLink souffrant du problème d'opérandes statiques multiplie par 2.1 le temps de transmission d'un mot de 32 bits. Les gains les plus importants s'effectuent au niveau des lectures.

5.3.2.2.Délais de lecture

Pour évaluer les latences logicielles en lecture pour le µBlaze et le PowerPC405, les délais de réception pour des messages dont la taille croît de 1 à 25 mots de 32 bits a été mesurée. La profondeur des FIFO matérielles est de 30 mots de 32 bits. On remarque que le comportement est linéaire (ANNEXE G). Quatre cas ont été étudiés :

- **Conventionnel Lecture** : Méthode sans DirectLink. Le message est lu d'une FIFO logicielle contenue dans le TOR et copié dans le tampon de réception.
- **Conventionnel Interruption** : Exécution de la routine d'interruption lorsque l'ISSAdapter notifie le µBlaze de la réception d'un message. Le message est copié de l'ISSAdapter dans une FIFO logicielle.
- **DirectLink** : Méthode utilisant le DirectLink. Les messages sont dé-sérialisés et lus de l'interface matérielle (FSL ou APU).
- **DirectLink Optimal** : Méthode utilisant le DirectLink avec des instructions pour lesquelles les opérandes sont fixes.

Tableau 5.4 – Temps d'exécution pour lecture sur µBlaze

Type	Préparation (cycles)	Transmission (cycles)	Interruption (cycles)
DirectLink	2424	400	2988
DirectLink Optimal	2293	165	2988
Conventionnel Interruption	7701	1632	
Conventionnel Lecture	14700	2944	
Conventionnel Total	22401	4576	

Dans le cas µBlaze, la méthode DirectLink est très avantageuse par rapport à la méthode conventionnelle de communication SPACE. Les temps de préparation diminuent de 90% avec le DirectLink. Comme pour l'écriture, ceci est du au fait qu'aucun en-tête de message ne doit être analysé et qu'une multitude de *sanity checks* sont évités. De plus, on évite la gestion des FIFO logicielles qui sont très lourdes à manipuler et beaucoup de changement de contexte.

Pour les temps de transmission, la méthode optimale DirectLink réduit de 96% le délai de transmission d'un mot de 32 bits et la méthode non optimale le réduit de 91%. Ce gain significatif est du au fait qu'on évite une double copie du message lors de sa manipulation.

Tableau 5.5 – Temps d'exécution pour lecture sur PowerPC405

Type	Préparation (cycles)	Transmission (cycles)	Interruption (cycles)
DirectLink	886	194	2241
DirectLink Optimal	882	49	2241
Conventionnel Interruption	4673	953	
Conventionnel Lecture	3068	836	
Conventionnel Total	7741	1789	

Sur le PowerPC405, les temps de préparation diminuent de 89% avec le DirectLink. Pour les temps de transmission, la méthode optimale DirectLink réduit de 97% le délai de transmission d'un mot de 32 bits et la méthode non optimale le réduit de 89%.

5.3.2.3.Comparaisons additionnelles

Sur les deux architectures, µBlaze et PowerPC, les gains relatifs obtenus pour l'utilisation du DirectLink sont cohérent et du même ordre de grandeur. Cependant, le PowerPC est plus rapide que le µBlaze pour effectuer les opérations de lecture et d'écriture logicielles. Ceci est principalement causé par le fait que l'assembleur généré

par le compilateur du PowerPC est beaucoup plus optimisé et que son jeu d'instruction est aussi plus efficace.

Outre le gain de performance obtenu avec le DirectLink par rapport à la méthode conventionnelle, deux autres aspects sont avantageux :

- **Support du débordement des FIFO matérielles :** DirectLink permet l'envoie de messages dont la taille dépasse la profondeur des FIFO matérielles. Même si cela dégrade les performances et que le processus qui transmet est bloqué, aucune donnée n'est perdue, l'intégrité des messages est préservée et le système continue de s'exécuter correctement.
- **Segmentation/Agrégation des messages :** DirectLink permet l'envoie séparé d'une multitude de petits paquets et leur lecture en agrégat, ou vice versa, i.e. l'envoie d'une mégastructure et sa lecture par petit paquets. Ce mécanisme est impossible dans la méthode conventionnelle puisque l'information sur chaque paquet est contenue dans un en-tête qui est inviolable. Cette caractéristique non prévue pourrait être utile pour par exemple :
    - Envoyer une matrice en entier pour augmenter le taux d'utilisation du lien et effectuer sa lecture par ligne pour sauver de la mémoire au niveau du tampon de lecture.
    - Envoyer chaque couleur (R-G-B) d'un pixel séparément dès que sa valeur est calculée et lire le pixel en entier pour effectuer un changement de domaine.

5.4. Impact sur l'utilisation des ressources matérielles

L'utilisation du DirectLink permet de réduire la complexité des adaptateurs de modules qui doivent gérer des communications bidirectionnelles par une seule interface, la gestion des en-têtes de messages, la transmission de messages ACK et les lourds

protocoles de bus partagés. Tous ces aspects sont éliminés avec le DirectLink. La seule chose qui persiste est les FIFO pour contenir les messages.

Puisqu'il n'existe pas d'outils d'estimation de surface dans SPACE et qu'aucun modèle VHDL n'a été développé, il est impossible de proposer une estimation valide de la surface sauvée pour les différents adaptateurs de protocoles DirectLink. Cependant, les ressources utilisées par le lien FSL et son *wrapper* – utilisés dans un DirectLink μBlaze/ μBlaze – sont connues (Tableau 5.6). Ces données peuvent donner une approximation de la réduction de surface obtenue avec DirectLink. De plus, le *wrapper* FSL ressemble au niveau complexité à l'adaptateur SDL-FSL. La surface utilisée par un adaptateur de module conventionnel (sans FIFO de réception) est connue et présentée dans le Tableau 5.7 [67].

Tableau 5.6 – Utilisation des ressources matérielles par un lien FSL (VIRTEX-2 Pro VP30)

Module	Blocs logiques	LUTs	Bascules (Flip flop)	F_{max} (MHz)
Lien FSL (taille 1)	21	3	36	303
Lien FSL (taille 512)	30	56	30	219
FSL Wrapper	65	120	34	209

Tableau 5.7 – Utilisation des ressources matérielles par un adaptateur de module (VIRTEX-2 Pro VP30)

	Blocs logiques	LUTs	Bascules (Flip flop)	F_{max} (MHz)
Adaptateur version 1	435	749	616	163
Adaptateur version 2	353	300	600	122

L'augmentation des ressources utilisées par un adaptateur de module en fonction du nombre de FIFO de réception (dépendant du nombre de modules avec qui il communique) est aussi connue (Tableau 5.8) [67].

Tableau 5.8 – Utilisation des ressources matérielles par un adaptateur de module en fonction du nombre de FIFO de réception (VIRTEX-2 Pro VP30)

Nb de FIFO de réception	Blocs logiques	LUTs	Bascules (Flip flop)	F_{max} (MHz)
1	681	871	1187	158
2	727	956	1259	158
3	843	1069	1450	158
4	923	1169	1582	157
5	1042	1266	1796	157

En partant de la prémisse qu'il faut deux liens FSL et un adaptateur pour générer un DirectLink, il est possible d'affirmer que le remplacement **complet** d'un adaptateur de module version 2 par un DirectLink permet de réduire (pour la portion purement contrôle du mécanisme, sans FIFO) de 70% le nombre de blocs logiques, de 58% le nombre de LUTs et de 82% le nombre de bascules. De plus, le mécanisme peut fonctionner à une fréquence d'horloge 72% plus élevée.

Dans le cas où l'élimination complète d'un adaptateur de module n'est pas possible (en conservant certaines communications via le canal de contrôle), le transfert d'un lien en DirectLink entraîne en moyenne (avec des FIFO de 512 éléments) une augmentation de 18% des blocs logiques, de 28% des LUTs mais une diminution de 30% des bascules.

Le DirectLink a aussi un impact direct sur la quantité et le type de mémoire utilisée. La quantité maximale de mémoire interne disponible sur un FPGA (BRAM) varie d'un modèle à l'autre : 306 kilooctets pour un Virtex-2 Pro VP30 et 756 kilooctets pour un Virtex-4 LX200. Or, la taille d'un binaire exécutable de logiciel applicatif SPACE pour microprocesseur peut facilement atteindre 300 kilooctets avec les pilotes, le RTOS et le TOR. Il est donc habituel que le programme ne puisse résider dans la mémoire BRAM du FPGA. Lorsque c'est le cas, le binaire exécutable est stocké dans une mémoire externe au FPGA, ce qui décuple les latences d'accès.

Le DirectLink permet de transférer les FIFO logicielles, très lentes et stockées en mémoire externe, en matériel utilisant la BRAM. Ceci accélère le traitement des données et évite la redondance et le gaspillage de tampons mémoire (FIFO d'envoi à l'intérieur de l'adaptateur de module, dans l'ISSAdapter et dans le TOR). De plus, puisque le DirectLink accélère la réception des messages au niveau logiciel, la profondeur des FIFO matérielles peut être moins élevée puisque moins de messages ont le temps de s'y accumuler. En effet, dans la méthode conventionnelle, certains modules producteurs sont beaucoup rapides que les modules consommateurs. Deux possibilités étaient envisageables pour éviter la perte de données [67] :

- Configurer une profondeur de FIFO de réception suffisamment grande, mais ayant le désavantages de consommer plus de BRAM.
- Utiliser le mécanisme d'écriture bloquante du bus OPB par lequel le module transmetteur doit maintenir sa requête d'écriture jusqu'à ce que le destinataire l'accepte (lorsque de l'espace de la FIFO se libère). Cette méthode a pour désavantage de réduire la fréquence de fonctionnement de l'adaptateur de module et du système car l'accès au bus est refusé aux autres modules désirant initier un transfert. La profondeur des FIFO peut cependant être réduite.

5.5. Accélération d'une application dans SPACE avec le DirectLink

Dans ses travaux de recherche [68], Cédric Migliorini a effectué une exploration des architectures de communication d'une application de décodage JPEG en se basant sur l'environnement SPACE, le processeur μBlaze et le bus OPB. Une des phases de sa méthodologie se base sur la réutilisation du DirectLink pour relier des modules ayant un fort débit de données entre eux. Son approche du DirectLink cadre dans une perspective purement utilisateur pour le développement d'applications embarquées.

Deux modules de son architecture – EX (*Extractor*) et IDCT (*Inverse Discrete Cosine Transform*) – ont été tour à tour placés dans la partition logicielle sur µBlaze. L'architecture de référence est multibus où tous les modules sont connectés au bus OPB.

Tableau 5.9 – Accélération d'une application JPEG avec DirectLink sur µBlaze

Configuration[1]	Description	Accélération[2]	Réduction du temps de simulation
Multibus 2e3	EX en logiciel, tout en DirectLink	2.75	2.51
Multibus 2f3	IDCT en logiciel, tout en DirectLink	5.71	4.80
Multibus 2g2	IDCT et EX en logiciel sur 2 processeurs différents, tout en DirectLink	5.65	4.73
Multibus 2h2	IDCT et EX en logiciel sur le même processeur, tout en DirectLink	1.74	1.32

[1] Cet identificateur réfère au nom de configuration dans les travaux de [68]
[2] Par rapport à l'architecture de référence multibus, basée sur le nombre de cycles d'exécution de l'application

Comme le reflète le Tableau 5.9, l'utilisation du DirectLink permet d'accélérer jusqu'à un facteur de 5.71 l'exécution d'une application comme le JPEG. Le gain dépend du choix des communications qui sont transférées en DirectLink. Plus le débit est élevé entre deux modules, plus ce choix est avantageux (IDCT par rapport à EX). Le fait de mettre deux modules en logiciel sur le même processeur (configuration Multibus 2h2), dégrade de beaucoup le gain atteignable. Ceci est causé par les changements de contextes nombreux entre les tâches logicielles.

Les gains par DirectLink s'expliquent facilement par la réduction des latences de communication : élimination des délais d'arbitrage du bus, élimination du phénomène de contention et de blocage du bus, élimination des acquittements d'écritures bloquantes, diminution du nombre d'interruptions sur les processeurs et accélération du traitement logiciel des communications. Le Tableau 5.9 montre aussi une réduction du temps de simulation du système avec l'utilisation du DirectLink. Ceci est causé par la diminution

du nombre de changements de contexte de l'ordonnanceur SystemC – causés par les processus d'arbitre et de temporisation du bus OPB – mais surtout par la réduction du temps d'exécution.

5.6. Extensibilité du paradigme à d'autres plateformes

La possibilité d'étendre le paradigme du DirectLink à d'autres plateformes a été étudiée pour deux cas, soit le XTensa de Tensilica et le NIOS-II d'Altera.

5.6.1. Tensilica XTensa

Le processeur XTensa de Tensilica est un processeur extensible qui offre encore plus de possibilités que le PowerPC405 FX.

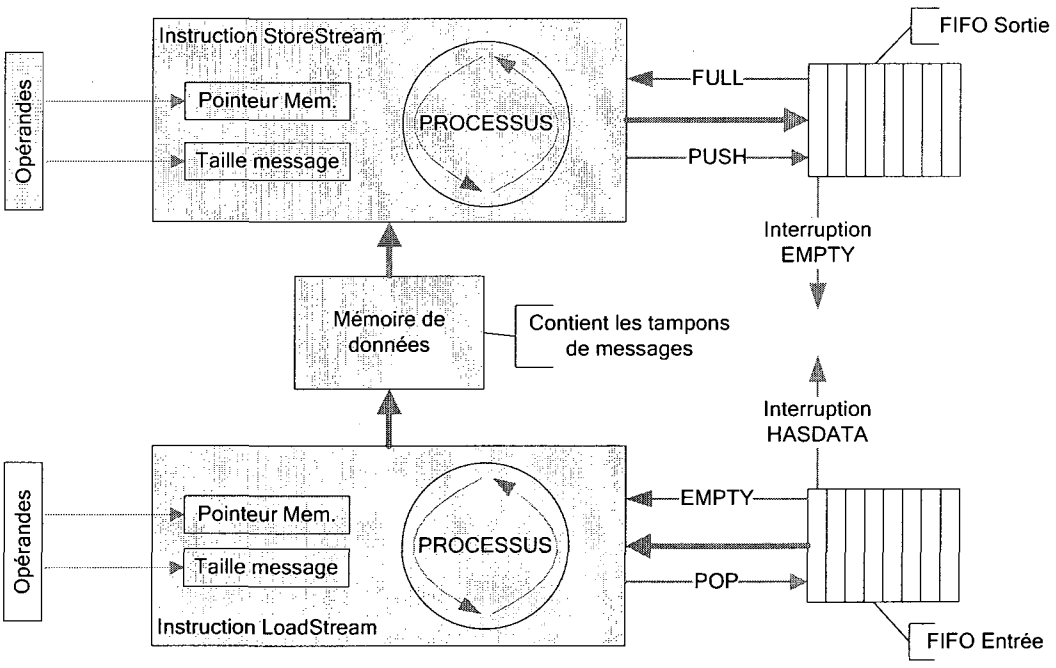


Figure 5.3 – Implémentation du DirectLink pour Tensilica Xtensa

Il permet l'ajout de nouvelles interfaces de communication et d'instructions totalement personnalisées à même le pipeline du processeur. Le mécanisme d'implémentation (Figure 5.3) du DirectLink pourrait être encore plus efficace que via l'APU.

Les opérandes des instructions sont l'adresse de base du tampon de message et la taille de celui-ci. Ces nouvelles unités d'exécution ayant leur propre interface à la mémoire peuvent envoyer/recevoir le message en entier via les FIFO en une seule instruction multicycle.

5.6.2. Altera NIOS-II

Comme le Tensilica Xtensa, le NIOS-II offre la possibilité d'ajouter de nouvelles interfaces de communication et des instructions totalement personnalisées à même le pipeline du processeur. Le mécanisme pourrait être exactement le même que celui de la Figure 5.3.

Cependant, le lien de communication standard du NIOS-II, le bus Avalon, est un *switch fabric* qui établit justement des liens point-à-point entre les périphériques matériels. Ce n'est pas un bus partagé comme le sont l'OPB et le PLB. Au moment du design de l'architecture, le concepteur doit préalablement établir ces canaux dédiés entre les différents composants à l'aide d'un outil graphique (Quartus). Même si la connexion en est une directe, le module est accessible via l'espace d'adressage du processeur.

Le DirectLink pourrait être utilisé comme paradigme de visualisation et d'établissement des connexions directes à l'intérieur de l'Avalon. Les données seraient stockées dans des FIFO à l'intérieur d'adaptateur de modules pour Avalon à la manière conventionnelle de SPACE. Le canal de contrôle du SpaceBaseModule deviendrait ici inutile.

5.7. Comparaison avec d'autres travaux

Une revue des travaux sur les méthodologies de communication à haut niveau pour les MPSoC a été présentée au chapitre 3. Dans le chapitre 4, l'approche du DirectLink est exposée pour implémenter en simulation (et possiblement sur FPGA) des liens point-à-point dans l'environnement SPACE.

Le DirectLink s'inscrit dans la tendance actuelle de reléguer au matériel une partie de la gestion logicielle des communications. Cependant, il s'attaque à certains problèmes rencontrés par les outils commerciaux lorsque vient le temps d'interconnecter différents processeurs hétérogènes et protocoles de communication [6] :

- Aucune hypothèse n'est faite sur le protocole de conversion au niveau des *wrappers* de module ni de la connectivité par défaut à un bus partagé. DirectLink permet la connection à de multiples composants via de multiples réseaux de communication utilisant des protocoles différents.
- DirectLink considère la génération automatique de l'architecture sous-jacente.

Le paradigme DirectLink est similaire au modèle développé pour ImpulseC (2.3.2) en ceci qu'il permet de relier à haut niveau différents processus par des FIFO synchrones ou asynchrones. La structure en couche de l'interface logiciel/matériel et l'abstraction des mécanismes de communication pour plusieurs architectures cible et niveaux d'abstraction est semblable au modèle proposé dans les travaux du professeur Jerraya (2.3.3). De plus, la sélection d'implémentation se base sur une librairie de composants disponibles. Cependant, contrairement à ces approches où chaque canal abstrait de communication doit être manipulé explicitement et individuellement, DirectLink propose une méthode de manipulation unifiée de toutes les interfaces d'un module, cache totalement les différents canaux disponible et favorise l'orthogonalité de la fonctionnalité de l'architecture de communication. De plus, DirectLink supporte un

mécanisme de passage de messages haut niveau, ce qui n'est pas le cas des méthodes mentionnées où seule la transmission de types de base est supportée. Le DirectLink généralise aussi une approche coprocesseur telle que proposée en [51] à toutes les classes de communication et en utilisant les mécanismes propres à chaque processeur.

Le DirectLink permet aussi de relier différents domaines d'horloges à l'aide de FIFO asynchrones à l'instar des travaux de [40]. La différence réside dans le fait qu'il n'est pas seulement possible de relier des sous-systèmes uniprocesseur mais bien chaque module ou chaque processeur.

Dans un autre ordre d'idées, l'ISS du PowerPC405 développé propose une nouvelle approche de génération basé sur la réutilisation des simulateurs inclus dans GNU GDB. C'est une méthodologie hybride de [24] et [25] – réutilisation de GNU GDB via RDI – et de [26] – instanciation de l'ISS comme classe C++ dans un modèle SystemC. Contrairement à [26], cet ISS intègre une amélioration proposée par [22] : combiner en un seul *thread* les éléments concurrents.

L'ISS du PowerPC pourrait tirer avantage d'une autre amélioration : intercepter certaines fonctions du noyau (*kernel*) pour les exécuter nativement sur le processeur hôte. En effet, puisque les mécanismes de communication DirectLink sont déterministe et que les latences sont connues, les appels aux primitives de communication pourraient être interceptées. De cette manière, les performances de la simulation seraient accrues sans perte de précision sur les temps d'exécution physiques.

Un des désavantages de la méthode de réutilisation de GNU GDB est la difficulté d'implémenter une vue architecturale du modèle de simulation de l'ISS. La méthode de [21] pourrait être implémentée sans introduire toute la complexité et la lenteur de la gestion des étages du pipeline par *threads* SystemC.

5.8. Améliorations suggérées à l'architecture de communication SPACE

La méthode conventionnelle de communication SPACE induit le problème d'atomicité des transferts des messages. La Figure 5.4 illustre le fait que lorsqu'un module 1 envoie un message vers un module 2, aucun autre module ne doit pouvoir communiquer avec le module 2. En effet, tous les modules envoient leurs messages à l'adresse de base des autres modules. Lors de la réception d'un message, le récipiendaire se fie à l'en-tête du message (premier mot reçu) pour diriger le reste des données dans la bonne FIFO. Si pendant un transfert, un second module envoie un message au même destinataire, les données seront traitées comme faisant partie du premier message. Ceci corrompt tous les transferts subséquents vers ce module.

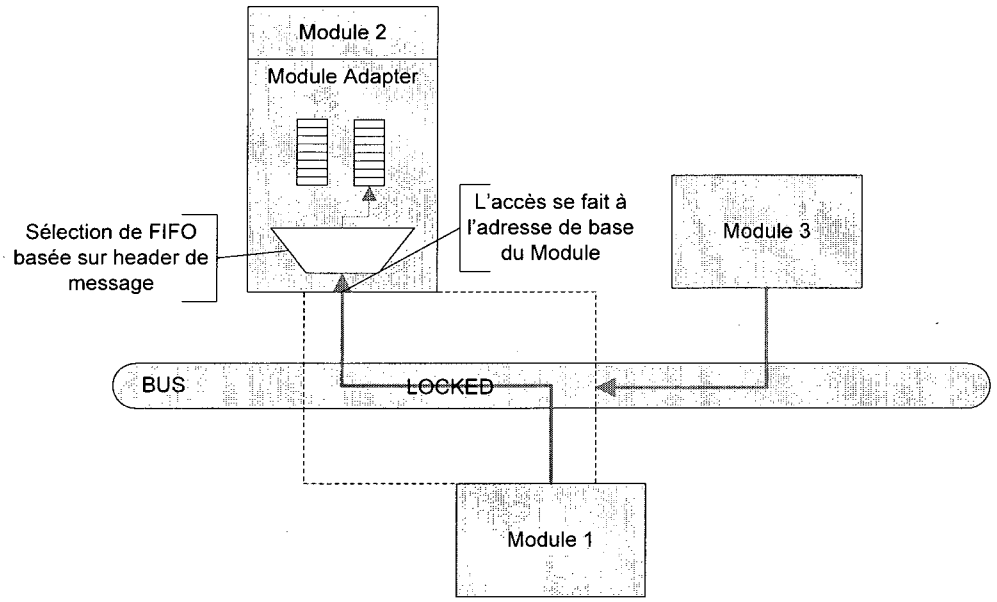


Figure 5.4 – Illustration du problème d'atomicité des transferts SPACE

Assurer l'atomicité peut devenir difficile. Le mode d'adressage des modules dépend de la présence d'un *bus lock* sur le bus pour garantir que les données d'un message soient associées au bon en-tête. De cette manière, aucun autre module ne peut initier un transfert (en lecture ou écriture) sur le bus partagé. Deux problèmes apparaissent :

- Le *bus lock* du PLB n'est pas accessible de manière logicielle (via une instruction ou un registre de contrôle) sur le PowerPC405. Ainsi, un module logiciel ne peut assurer l'atomicité de ses transferts vers le matériel.
- Dans le cas de l'OPB, lorsqu'un module logiciel s'exécutant sur un μBlaze transfert un message, il bloque les transmissions de tous les autres modules matériels sur le bus. Or, comme vu à la section 5.3.2, un transfert logiciel peut prendre quelques milliers de cycles et pose un faible taux d'utilisation du bus. Ceci devient problématique si l'application a besoin de beaucoup de bande passante.

Cette problématique s'ajoute à une autre faiblesse du protocole de communication SPACE : l'allocation de 4 mégaoctets d'espace d'adresse par module utilisateur, alors que seule l'adresse de base est utilisée pour les transmissions de message.

Une solution est donc proposée (Figure 5.5) : au lieu d'envoyer tous les messages vers l'adresse de base d'un module destinataire, ils sont plutôt envoyés à un décalage correspondant à leur identificateur numérique SPACE. Si le protocole est respecté et puisque tous les identificateurs sont uniques, il n'est plus nécessaire de bloquer le bus lors d'un transfert. Les données sont dirigées vers la bonne FIFO en fonction de l'adresse de réception des messages. Ainsi, il est possible d'entrelacer les transmissions sur le bus, d'augmenter la bande passante et de réduire le contrôle au niveau logiciel.

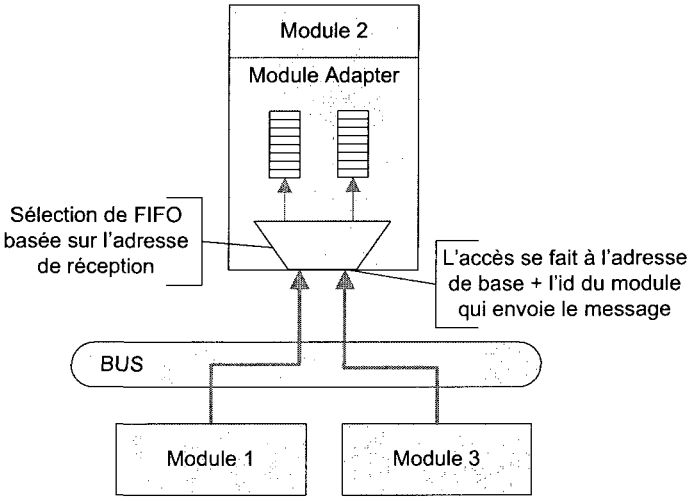


Figure 5.5 – Résolution du problème d'atomicité des transferts

L'implémentation de cette solution ne requérait que la modification des adaptateurs de module ainsi que de certaines fonctions des couches inférieures du HAL pour la gestion des communications.

CONCLUSION ET TRAVAUX FUTURS

L'état actuel des méthodologies de conception de MPSoC vise à intégrer verticalement et à haut niveau la conception, la simulation et la validation des partitions logicielles et matérielles d'une architecture. Ce nouveau flot raffine progressivement les modules et l'architecture de communication jusqu'à l'atteinte des objectifs de performances.

La plateforme de développement Space, basée sur la librairie SystemC, permettant la validation, l'exploration architecturale et le partitionnement logiciel-matériel à haut niveau d'un MPSoC a été décrite. Elle fournit un protocole de communication basé sur l'échange de message via un bus partagé entre les différents modules d'une architecture TLM.

Les travaux de recherche actuels ciblant la conception de MPSoC au niveau TLM tentent d'unifier le concept d'interface matériel/logiciel mais ne proposent pas de méthodologie pour gérer plusieurs ports et protocoles de communication différents avec une seule et même interface, laissant au concepteur le choix d'ajouter ou de retirer des canaux sans affecter la description fonctionnelle des modules.

S'inscrivant dans ce courant de pensée, ce projet a introduit le DirectLink, soit un paradigme d'abstraction des communications point-à-point basé passage de message et s'intégrant à l'environnement Space. De plus, DirectLink fait usage des nouvelles opportunités qu'offrent les processeurs embarqués tels les canaux haut-débit et l'extensibilité du jeu d'instruction. La plateforme Space est modifiée pour n'offrir qu'un seul point d'entrée pour toutes les communications et un routage automatique des messages.

DirectLink améliore les performances des communications en restant indépendant de la plateforme matérielle cible. Les délais induits par les protocoles standards SPACE sont réduis d'environ 90% et l'exécution d'une application complexe peut être accélérée par un facteur 5.

Travaux futurs

Certains travaux sont envisageables pour améliorer encore DirectLink et la plateforme Space. Tout d'abord le nouveau mécanisme de communication pour les bus partagé proposé dans ce projet devrait être implémenté pour éliminer le blocage du bus, augmenter son taux d'utilisation et favoriser son extensibilité à d'autres architectures.

DirectLink et l'intégration d'un nouvel ISS dans SPACE ouvrent la porte à de nombreuses opportunités :

- Étudier la génération automatique des interconnexions de composants à Simtek en ne se basant pas sur une librairie prédéfinie d'implémentations disponibles. La méthode actuelle requiert du développeur de manuellement décrire quels modules sont nécessaires et comment ils doivent être connectés ensembles. Le choix devrait se faire de manière algorithmique et se baser sur une description des interfaces à interconnecter. Les travaux de Nicolas Laug du CIRCUS vont en ce sens.
- Intégrer la gestion de domaines d'horloges multiples afin d'explorer la conception de GALS dans Space et de permettre une estimation et une minimisation à haut niveau de la puissance consommée par une architecture. La plateforme devrait être capable de maximiser le ratio performance/puissance automatiquement.
- Explorer la conception de systèmes multiprocesseurs hétérogènes et établir une méthodologie de partitionnement qui permet de sélectionner quel type de

processeur est plus apte à exécuter une tâche logicielle dans l'objectif global de maximiser le ratio performance/puissance automatiquement.

- Étudier la génération et l'intégration dans Space de simulateurs d'instruction multi-niveau d'abstraction pour des processeurs extensibles. Une des difficultés actuelle est d'intégrer différentes vues de modélisation dans l'ISS afin de permettre un réglage de sa précision d'exécution. L'autre problème est l'ajout à l'ISS, lors de la modélisation d'un système, d'instructions spécialisées définies par l'usager et de nouvelles interfaces de communication. L'ISS du PowerPC405 développé par IBM ne permet pas, par exemple, la modélisation de l'APU. Cette nouvelle méthodologie devrait être générique pour faciliter l'ajout de plusieurs nouveaux processeurs à Space.
- Établir une méthodologie de partitionnement automatique à 3 choix : logiciel, matériel partagé et matériel dédié. L'objectif est de trouver le bon compromis puissance/performance/surface/flexibilité/réutilisabilité.

RÉFÉRENCES

[1] Netrino. Embedded Systems Glossary [En ligne], 2008. www.netrino.com/Embedded-Systems/Glossary/ (page consultée le 8 janvier 2008)

[2] Bass, M.J.; Christensen, C.M., "The future of the microprocessor business," *IEEE Spectrum*, vol.39, no.4, pp.34-39, Apr. 2002.

[3] Electronics.ca Research Network. High Growth Expected in the Worldwide Embedded System Market in the Next Five Years. [En ligne]. http://www.electronics.ca/presscenter/articles/95/1/ (Page consultée le 19 février 2008).

[4] Mak, T.S.T.; Sedcole, P.; Cheung, P.Y.K.; Luk, W., "On-FPGA Communication Architectures and Design Factors," *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on* , vol., no., pp.1-8, 28-30 Aug. 2006

[5] J. Chevalier, M. de Nanclas, L. Filion, O. Benny, M. Rondonneau, G. Bois and E.M. Aboulhamid. A SystemC Refinement Methodology for Embedded Software. *Design and Test of Computers,* IEEE, 23(2):148-158, 2006.

[6] JERRAYA, A.A., WOLF, W., *« Multiprocessor Systems-on-Chip »*, Morgan Kaufmann, 608 pages, 2004.

[7] CoreConnect™ bus architecture (White Paper), 1999. [En ligne]. http://www-306.ibm.com/chips/products/coreconnect/. (Pages consultées 15 janvier 2008).

[8] ARM AMBA Specification. [En ligne]. http://www.arm.com/products/solutions/AMBA_Spec.html. (Page consultée le 20 janvier 2008).

[9] Avalon Memory-Mapped Interface Specification. [En ligne]. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf. (Page consultée le 20 janvier 2008).

[10] KANGAS Tero. Methods and Implementations for Automated System On Chip Architecture Exploration. Tampere University of Technology. Publication 616. 29th of September 2006.

[11] A. A. Jerraya and W. Wolf. Hardware/Sofware Interface Codesign for Embedded Systems. *Computer*, 38(2):63-69, 2005.

[12] Serra, M., Gardner, W.B., Hardware/Software Codesign: Introducing an interdisciplinary course [En ligne], 1998. http://www.cs.ubc.ca/wccce/program98/micaela/micaela.html (Pages consultées 15 janvier 2008).

[13] Cai, L. and Gajski, D. 2003. Transaction level modeling: an overview. In *Proceedings of the 1st IEEE/ACM/IFIP international Conference on Hardware/Software Codesign and System Synthesis* (Newport Beach, CA, USA, October 01 - 03, 2003). CODES+ISSS '03. ACM, New York, NY, 19-24.

[14] S. Pasricha. Transaction Level Modelling of SoC with SystemC 2.0. In Synopsys User Group Conference, 2002.

[15] PELLERIN, D., THIBAULT, S., *« Practical FPGA programming in C »*, Prentice Hall PTR, 464 pages, 2005.

[16] SystemC. SystemC User's Guide [En ligne], 2008. www.systemc.org. (Pages consultées 15 janvier 2008).

[17] IEEE. IEEE Std 1666 – 2005 IEEE Standard SystemC Language Reference. Technical report, IEEE, 2005.

[18] Introduction à SystemC. [En ligne]. http://comelec.enst.fr/hdl/sc_intro.html. (Page consultée le 19 février 2008).

[19] Instruction Set Simulator, Article Wikipedia [En ligne], 2008. http://en.wikipedia.org/wiki/Instruction_Set_Simulator. (Pages consultées 15 janvier 2008).

[20] Lu, Yen-Ju; Lin, Chen-Tung; Wu, Chi-Feng; Hwang, Shih-Arn; Lin, Ying-Hsi, "Microprocessor Modeling and Simulation with SystemC," *VLSI Design,*

Automation and Test, 2007. VLSI-DAT 2007. International Symposium on , vol., no., pp.1-4, 25-27 April 2007

[21] In-Cheol Park; Sehyeon Kang; Yongseok Yi, "Fast cycle-accurate behavioral simulation for pipelined processors using early pipeline evaluation," *Computer Aided Design, 2003. ICCAD-2003. International Conference on* , vol., no., pp. 138-141, 9-13 Nov. 2003

[22] Rissa, T.; Donlin, A.; Luk, W., "Evaluation of SystemC modelling of reconfigurable embedded systems," *Design, Automation and Test in Europe, 2005. Proceedings* , vol., no., pp. 253-258 Vol. 3, 7-11 March 2005

[23] Oussorov, I.; Raab, W.; Hachmann, U.; Kravtsov, A., "Integration of instruction set simulators into SystemC high level models," *Digital System Design, 2002. Proceedings. Euromicro Symposium on* , vol., no., pp. 126-129, 2002

[24] Fummi, F.; Martini, S.; Perbellini, G.; Poncino, M., "Native ISS-SystemC integration for the co-simulation of multi-processor SoC," *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings* , vol.1, no., pp. 564-569 Vol.1, 16-20 Feb. 2004

[25] Formaggio, L.; Fummi, F.; Pravadelli, G., "A timing-accurate HW/SW cosimulation of an ISS with SystemC," *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on* , vol., no., pp. 152-157, 8-10 Sept. 2004

[26] Benini, L.; Bertozzi, D.; Bruni, D.; Drago, N.; Fummi, F.; Poncino, M., "SystemC cosimulation and emulation of multiprocessor SoC designs," *Computer* , vol.36, no.4, pp. 53-59, April 2003

[27] FILION Luc, CANTIN Marc-André, MOSS Laurent, ABOULHAMID Mostapha, BOIS Guy. Space Codesign: A SystemC Framework for Fast Exploration of Hardware/Software Systems. *Proceedings of the Design & Verification Conference and Exhibition (DVCON'07)*, février 2007, 8 p.

[28] LEIBSON, S., « Designing SOCs with Configured Cores », Morgan Kaufman, 2006.

[29] J. Henkel. Closing the SoC design gap. *Computer*, 36(9):119-121, 2003.

[30] Nurmi, J.; Leibson, S.; Campi, F.; Panis, C., "Extensible and Configurable Processors for System-on-Chip Design," *Advanced Signal Processing, Circuits, and System Design Techniques for Communications, 2006* , vol., no., pp.45-97, May 2006

[31] LABROSSE, Jean. J. *MicroC/OS-II: The Real Time Kernel.* Newnes, 648 pages, 2002.

[32] ROWEN Chris. Engineering the Complex SOC. Prentice Hall Modern Semiconductor Design Series. ISBN 0-13-145537-0, 2004

[33] Mak, T.S.T.; Sedcole, P.; Cheung, P.Y.K.; Luk, W., "On-FPGA Communication Architectures and Design Factors," *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on* , vol., no., pp.1-8, 28-30 Aug. 2006

[34] ARM, "AMBA Multi-layer AHB overview", Tech. Rep., 2001.

[35] D. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*, Ph.D. Thesis, Stanford University, 1984.

[36] Kishinevsky, M.; Shukla, S.K.; Stevens, K. S., "Guest Editors' Introduction: GALS Design and Validation," *IEEE Design & Test of Computers* , vol.24, no.5, pp.414-416, Sept.-Oct. 2007

[37] Teehan, P.; Greenstreet, M.; Lemieux, G., "A Survey and Taxonomy of GALS Design Styles," *IEEE Design & Test of Computers* , vol.24, no.5, pp.418-428, Sept.-Oct. 2007

[38] KNiyogi, D. Marculescu, "Speed and Voltage Selection for GALS Systems based on Voltage Frequency Islands," Proceedings of the ACM-IEEE Asia-Pacific Design Automation Conference," China, January *2005.*

[39] Hemani, A.; Meincke, T.; Kumar, S.; Postula, A.; Olsson, T.; Nilsson, P.; Oberg, J.; Ellervee, P.; Lundqvist, D., "Lowering power consumption in clock by using

globally asynchronous locally synchronous design style," *Design Automation Conference, 1999. Proceedings. 36th* , vol., no., pp.873-878, 1999

[40] Smith, S.F., "An asynchronous GALS interface with applications," *Microelectronics and Electron Devices, 2004 IEEE Workshop on* , vol., no., pp. 41-44, 2004

[41] Niyogi, K.; Marculescu, D., "System level power and performance modeling of GALS point-to-point communication interfaces," *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on* , vol., no., pp. 381-386, 8-10 Aug. 2005

[42] Hoare, C.A.R. *Communication Sequential Processes.* Upper Saddle River, NJ: Prentice Hall, 1985.

[43] Shenoy, Kunal, Accelerating Software Applications Using the APU Controller and C-to-HDL Tools. Xilinx Application Note XAPP901, Xilinx, Inc., 2005.

[44] Shenoy, Kunal, Implementing a Virtex-4 FX PowerPC System with a C-to-HDL Hardware Coprocessor Accelerator. Xilinx, Inc., 2005.

[45] Daveau, J.-M.; Marchioro, G.F.; Ben-Ismail, T.; Jerraya, A.A., "Protocol selection and interface generation for HW-SW codesign," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.5, no.1, pp.136-144, Mar 1997

[46] Popovici, Katalin; Guerin, Xavier; Brisolara, Lisane; Jerraya, Ahmed, "Mixed Hardware Software Multilevel Modeling and Simulation for Multithreaded Heterogeneous MPSoC," *VLSI Design, Automation and Test, 2007. VLSI-DAT 2007. International Symposium on* , vol., no., pp.1-4, 25-27 April 2007

[47] Bouchhima, A.; Bacivarovx, I.; Youssef, W.; Bonaciu, M.; Jerraya, A.A., "Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration," *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific* , vol.2, no., pp. 969-972 Vol. 2, 18-21 Jan. 2005

[48] Guerin, Xavier; Popovici, Katalin; Youssef, Wassim; Rousseau, Frederic; Jerraya, Ahmed, "Flexible Application Software Generation for Heterogeneous Multi-

Processor System-on-Chip," *Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 1. 31st Annual International* , vol.1, no., pp.279-286, 24-27 July 2007

[49] Aimen Bouchhima; Lobna Kriaa; Wassim Youssef; Patrice Gerin; Frederic Petrot; Ahmed A. Jerraya, "A Unified HW/SW Interface Refinement Approach for MPSoC Design," *Circuits and Systems, 2006 IEEE North-East Workshop on* , vol., no., pp.185-188, June 2006

[50] Daveau, J.-M.; Marchioro, G.F.; Ben-Ismail, T.; Jerraya, A.A., "Protocol selection and interface generation for HW-SW codesign," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.5, no.1, pp.136-144, Mar 1997

[51] Singh, A.; Chhabra, A.; Gangwar, A.; Dwivedi, B.K.; Balakrishnan, M.; Kumar, A., "SoC synthesis with automatic hardware-software interface generation," *VLSI Design, 2003. Proceedings. 16th International Conference on* , vol., no., pp. 585-590, 4-8 Jan. 2003

[52] Leibson, S.; Kim, J., "Configurable processors: a new era in chip design," *Computer*, vol.38, no.7, pp. 51-59, July 2005.

[53] Shee, Seng Lin; Parameswaran, "Design Methodology for Pipelined Heterogeneous Multiprocessor System," *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE* , vol., no., pp.811-816, 4-8 June 2007

[54] IBM, (2007). Processor Local Bus (128 bits), Specifications. [En ligne]. www.ibm.com. (Pages consultées 15 janvier 2008).

[55] IBM, (2004). On-Chip Peripheral Bus, Specifications. [En ligne]. www.ibm.com. (Pages consultées 15 janvier 2008).

[56] IBM, (2006). Device Control Register Bus 3.5, Architecture Specifications. [En ligne]. www.ibm.com. (Pages consultées 15 janvier 2008).

[57] Xilinx, (2005). Fast Simplex Link (FSL) Bus (v2.00a). Rapport technique.

[58] Xilinx, (2007). μBlaze Processor Reference Guide. UG081 (v8.0). [En ligne]. www.xilinx.com. (Pages consultées 15 janvier 2008).

[59] The PowerPC 405 Core, White Paper, January 11, 1998. [En ligne]. www.ibm.com. (Pages consultées 15 janvier 2008).

[60] PowerPC 405 Processor Block Reference Guide: Embedded Development Kit, User Guide UG018, July 20, 2005. [En ligne]. www.xilinx.com. (Pages consultées 15 janvier 2008).

[61] PowerPC Instruction Set Extension Guide: ISA Support for the PowerPC APU Controller in Virtex-4, EDK, Document Revision 2.0, April 28, 2005. [En ligne]. www.xilinx.com. (Pages consultées 15 janvier 2008).

[62] PSIM : Emulator of the PowerPC Architecture. [En ligne]. http://sourceware.org/psim/manual/, (Page consultée le 30 janvier 2008).

[63] Cathy May , Ed Silha , Rick Simpson , Hank Warren , CORPORATE International Business Machines, Inc., The PowerPC architecture: a specification for a new family of RISC processors, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1994

[64] BOOK E – Enhenced PowerPC Architecture. [En ligne]. http://www-01.ibm.com/chips/techlib/. (Page consultée le 20 janvier 2008).

[65] The IBM PowerPC Embedded Environment: Architectural Specifications for IBM PowerPC Embedded Controllers, Specifications, v1.0. [En ligne]. www.ibm.com. (Pages consultées 15 janvier 2008).

[66] PowerPC Processor Reference Guide, User Guide UG011 (v1.2), January 19, 2007. [En ligne]. www.xilinx.com. (Pages consultées 15 janvier 2008).

[67] FAIZ, A. Implémentation des communications d'une plateforme SystemC sur un système reprogrammable de type FPGA. Mémoire de maîtrise, Département de Génie Informatique, École Polytechnique de Montréal, 2007.

[68] MIGLIORINI, C. *Exploration architecturale de communication-sur-puce au niveau système*, Mémoire de maîtrise, Département de Génie Informatique, École Polytechnique de Montréal, 2008.

[69] ALTERA, NIOS-II Processor Reference Handbook, Document Revision 7.2, October 2007. [En ligne]. www.altera.com. (Pages consultées 15 janvier 2008).

[70] LAUG, N. *Gestion générique et raffinement de systèmes électroniques à haut niveau*, Mémoire de maîtrise, Département de Génie Informatique, École Polytechnique de Montréal, Parution prévue pour décembre 2008.

[71] MOSS, L. *Profilage, caractérisation et partitionnement fonctionnel dans une plate-forme de conception de systèmes embarqués*, Thèse de doctorat, Département de Génie Informatique, École Polytechnique de Montréal, Parution prévue pour décembre 2009.

[72] Moss, L.; Cantin, M.-A.; Bois, G.; Aboulhamid, M., "Automation of Communication Refinement and Hardware Synthesis within a System-Level Design Methodology", *Rapid System Prototyping. 19th International Symposium on,* Monterey, 2-5 Jun. 2008. À paraître.

[73] Goyette S., Filion L., Migliorini C., (2007) Space Codesign: Spécification du lien point à point (v1.0.1). Rapport Technique.

ANNEXE A

MPSOC : DÉFINITION, PROBLÉMATIQUE ET CONCEPTION

A.1. Définition des MPSoC

Avant toute chose, il est important de définir ce qu'est un SoC multiprocesseur, concept qui sera référé ultérieurement sous l'appellation MPSoC. Un SoC est tout d'abord un circuit intégré dont la principale caractéristique est la complexité et qui implémente la majorité ou l'ensemble d'un système électronique complet. Les SoC sont principalement constitués de logique numérique et peuvent contenir de la mémoire, un processeur à jeu d'instruction, des bus et de la logique spécialisée. Un MPSoC est tout simplement un SoC qui contient plusieurs processeurs à jeu d'instruction. La conception du logiciel s'exécutant sur ces processeurs est un aspect important de la conception globale [6].

Bien que les systèmes multiprocesseurs traditionnels, telles les grilles de calcul, soient un sujet d'étude depuis fort longtemps en informatique, les MPSoC diffèrent en 4 points majeurs :

- Ils doivent répondre à des contraintes de temps réel ;
- Ils doivent utiliser une surface limitée sur un circuit ;
- Leur consommation énergétique doit être efficace ;
- Ils doivent gérer des entrées-sorties spécifiques.

A.2. Difficulté de la conception au niveau système

La conception d'un MPSoC est une tâche ardue qui comporte de nombreux aspects. Cette section identifie et décrit les difficultés majeures pertinentes au travail effectué afin de fournir un aperçu des enjeux impliqués dans le design.

A.2.1. Partitionnement logiciel/matériel

Un des aspects complexe de la conception des SoC est le partitionnement qui consiste à séparer un système en deux parties : le logiciel et le matériel. Ces deux parties nommées partitions sont décidées à l'avance et permettent d'atteindre l'orthogonalité de la synthèse du logiciel et du matériel. La synthèse du logiciel implique le développement des applications qui s'exécuteront sur les processeurs à jeux d'instruction. La synthèse matérielle implique quant à elle l'élaboration d'une plateforme qui permettra l'exécution de la partition logicielle.

Le partitionnement est une étape importante car c'est le processus à travers lequel le système sur puce est conçu pour qu'il réponde aux requis fonctionnels et de performances. Une division adéquate et optimale des fonctionnalités entre les deux partitions pour répondre aux différentes contraintes du système se révèle être un problème NP-complet.

Typiquement, les contraintes à respecter lors du partitionnement sont la consommation de mémoire, la surface requise, la bande passante, le temps d'exécution, la puissance dissipée et les coûts de développement. Le compromis coût/performance peut s'avérer primordial lors du design, i.e. un partitionnement impliquant une couche logicielle plus importante coûte moins cher à produire mais est nécessairement moins performant qu'un partitionnement où les mêmes fonctionnalités sont implantées en matériel. L'utilisation de processeurs extensibles devient dans ce contexte une solution intéressante.

A.2.2. Architectures de communication

Une fois le partitionnement des fonctionnalités en composants matériel et logiciels effectué, une majeure partie de la tâche à réaliser est l'organisation des interconnexions

entre ces composants. L'impact d'une architecture de communication pour un SoC est important sur les plans de la surface utilisée, des performances globales et de la dissipation de puissance. Les choix de conception font légions. Traditionnellement basés sur les bus partagés, les SoC voient depuis quelques années l'apparition de nouvelles topologies de communications : réseaux-sur-puces, GALS, liens point-à-point, etc.

Afin de contrer la complexité grandissante des MPSoC, différentes architectures standard ont été développées comme la plateforme CoreConnect [7] d'IBM, AMBA [8] de ARM ou Avalon [9] d'Altera.

A.2.3. Hétérogénéité des systèmes

De plus, du fait que la plupart des MPSoCs soient hétérogènes les rendent plus difficiles à programmer que dans un contexte plus traditionnel de multiprocesseur symétrique. De plus, l'agglomération sur une même puce de composants hétérogène augmente la complexité du flot de conception, surtout au niveau du réseau d'interconnexions. Chaque processeur peut posséder une interface distincte et doit quand même être capable de communiquer avec les autres composants.

A.3. Conception des MPSoC

Dans [10] et [11], l'approche traditionnelle de conception des SoC passe par 4 phases :

- Conception de l'architecture ;
- Conception des blocs logiciels et matériels ;
- Intégration du matériel ;
- Implémentation du logiciel.

La conception de l'architecture comporte l'étape d'exploration architecturale qui tente d'identifier un partitionnement idéal pour les contraintes du système en se basant sur l'expérience des concepteurs. Une fois le partitionnement effectué, le développement parallèle du matériel et logiciel peut débuter en se basant sur les spécifications fonctionnelles initiales. Cette approche présente de nombreuses faiblesses :

- Le concepteur tente traditionnellement de maximiser la portion logicielle afin d'avoir plus de flexibilité. Il peut devenir alors coûteux de modifier le partitionnement pour transférer des modules en matériel si les performances ne sont pas au rendez-vous.
- La spécification ne permet pas une représentation unifiée du logiciel et du matériel rendant la validation globale du système très difficile.
- Le débogage des interfaces logiciel-matériel se complexifie.
- La complexité des architectures affecte considérablement la durée du cycle de développement des systèmes.
- Le partitionnement choisi a une très forte probabilité d'être sous-optimal.

A.3.1. Codesign logiciel/matériel

Ces faiblesses tentent d'être palliées par de nouveaux processus de conception. De plus, le choix du partitionnement devrait se baser sur des métriques et les caractéristiques propres au système en cours de conception. La conception conjointe logiciel/matériel (Figure A.1), communément appelée codesign, tente d'y répondre. L'objectif est d'être capable de développer en parallèle tous les aspects d'une architecture, des spécifications jusqu'au prototype.

Cette nouvelle méthodologie peut se définir ainsi [12]:

- La conception coopérative des composants matériels et logiciels ;
- L'unification des flots de conception matérielle et logicielle ;

- Le mouvement des fonctionnalités entre le logiciel et le matériel ;
- La rencontre des objectifs de niveau système en exploitant la synergie du matériel et du logiciel à travers leur conception concurrente.

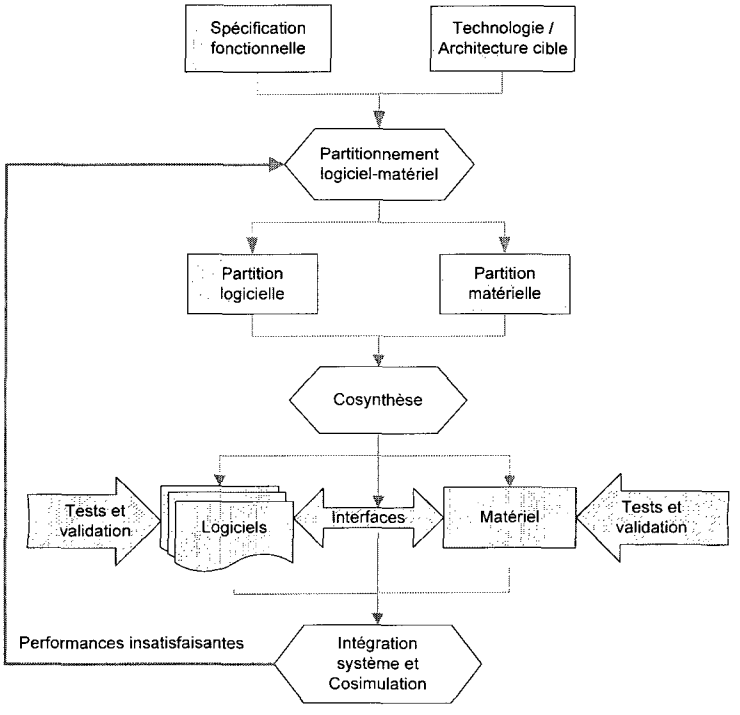


Figure A.1 – Méthodologie de codesign

Dans cette nouvelle méthodologie, le système est spécifié à haut niveau en utilisant un langage comme le C, C++ ou le SystemC. Le système est simulé à haut niveau dans un environnement de cosimulation en intégrant souvent le modèle TLM et la modélisation s'insère dans un processus itératif visant à établir un partitionnement idéal. L'interaction entre le logiciel et le matériel dès les premières phases de conception permet une rétroaction plus rapide sur les choix de design initiaux.

A.3.2. Conception au niveau système

La conception TLM (*Transaction Level Modeling*) a pour but d'augmenter d'un cran le niveau d'abstraction des SoC. Le concept est de modéliser uniquement les échanges de données qui occurrent entre les composants d'une architecture. Le détail des signaux matériels est éliminé pour ne mettre l'emphase que sur le comportement du système via ses transferts de données. La communication est séparée des calculs et est modélisée par des canaux. Une transaction s'effectue via l'appel de fonctions offertes par les interfaces des canaux [13].

De manière générale, le TLM offre trois niveaux d'abstraction :

- Validation de la fonctionnalité sans notions temporelles ;
- Intégration de notions temporelles pour la simulation de délais ;
- Simulation au cycle près.

Dans ce contexte, une transaction représente un échange de donnée ou un événement entre deux composants d'une simulation. Le protocole est secondaire puisque la microarchitecture n'est pas vérifiée. Une transaction de donnée peut être un mot, une série de mots ou une structure de donnée complexe qui est transférée sur un bus. Une transaction événementielle modélise les aspects de synchronisation assurant l'opération correcte du SoC. Une interruption en est un exemple [14].

ANNEXE B

SYSTEMC : LANGAGE DE DESIGN AU NIVEAU SYSTÈME

Dans un flot de conception au niveau système, la fonctionnalité d'une architecture est capturée dans un langage de description approprié. Les langages HandelC, SpecC, HardwareC, ImpulseC [15], SystemVerilog et SystemC [16] sont les plus connus et répandus. L'emphase sera ici mise sur SystemC qui est au cœur des présents travaux de recherche.

SystemC est une bibliothèque de classes développée en C++. Cette librairie permet une modélisation à haut niveau (TLM) de composants matériels et de concepts habituellement retrouvés dans les langages de description matériel (VHDL, Verilog, etc.) comme les signaux, les événements discrets, les listes de sensibilités, les horloges, etc. Le modèle de calcul utilisé est le même que celui de VHDL et Verilog.

Le standard IEEE 1666 [17] uniformise les pratiques d'utilisation de SystemC pour effectuer une modélisation système au niveau transactionnel (TLM). Il est à noter que SystemC, dans sa forme actuelle, n'offre qu'un support très limité de la modélisation du logiciel. SystemC est concrètement une collection de classes représentants des concepts matériels ainsi qu'un simulateur événementiel.

B.1. Composants architecturaux

La Figure représente l'architecture en couche de SystemC. Le lecteur peut se référer à [16] pour obtenir des détails sur chaque composante.

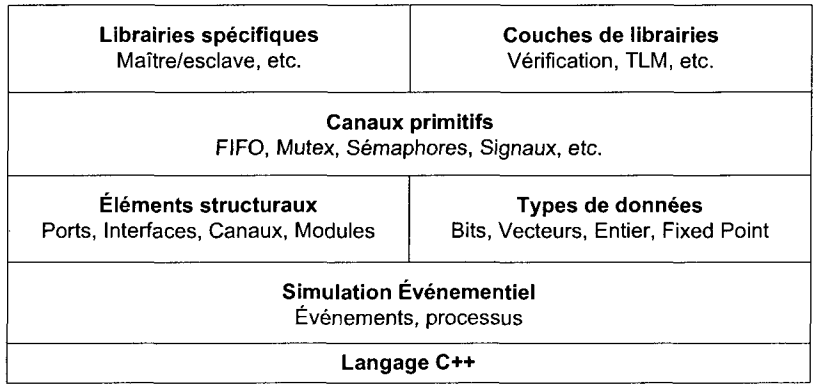


Figure B.1 – Architecture de SystemC

Les éléments structuraux permettant de modéliser une architecture de MPSoC sont les suivants :

- **Module :** Un module représente généralement un bloc réalisant une fonctionnalité précise du système. C'est une classe C++ décrivant le comportement du bloc, qui contient un ou plusieurs processus légers (*thread*), des ports de communication, des sous-modules et des données et qui peut implémenter une ou plusieurs interfaces.
- **Interface :** C'est une agrégation de méthodes virtuelles qui peuvent être utilisées à travers un port SystemC.
- **Ports :** Ce sont des objets C++ qui permettent de communiquer avec d'autres composants architecturaux via un contrat préétabli par la définition d'une interface.
- **Canaux :** Ce sont des objets de communication entre les modules d'une architecture. Ils se connectent sur les ports des modules et implémente les interfaces définies. Les canaux peuvent être hiérarchiques afin de cacher la complexité de certains média de communication.

B.2. Niveaux de raffinement

SystemC offre plusieurs niveaux comportementaux afin de modéliser des architectures avec différents niveaux de détails sur la fonctionnalité. Il est ainsi possible de passer à une spécification au niveau TLM à une implémentation RTL. Les différents niveaux supportés sont les suivants [18] :

- **BCA (Bus Cycle Accurate)** : s'applique à l'interface d'un modèle. Il signifie que la modélisation des transactions sur l'interface est correcte au cycle près.
- **BA (Bit Accurate)** : s'applique à l'interface d'un modèle. Il signifie que la modélisation des transactions sur l'interface est BCA, et porte aussi sur les signaux de l'interface. La modélisation est précise au bit près.
- **UTF (UnTimed Functional)** : s'applique à l'interface et à la fonctionnalité d'un modèle. Le modèle ne comporte aucune notion de durée d'exécution, mais seulement un ordre éventuel dans l'exécution des événements.
- **TF (Time Functional)** : s'applique à l'interface et à la fonctionnalité d'un modèle. Le modèle comporte des notions de durée (temps d'exécution des processus, latence, temps de propagation, etc.)
- **RTL (Register Transfert Level)** : s'applique à l'interface et à la fonctionnalité d'un modèle matériel. Chaque bit, chaque cycle, chaque registre du système est modélisé.

ANNEXE C

ISS : OUTIL DE COSIMULATION

Durant la cosimulation, les blocs logiciels sont exécutés sur un ou plusieurs simulateurs de jeu d'instructions, ou « Instruction Set Simulator » (ISS), selon le nombre de processeurs dans l'architecture. Ce simulateur interagit avec les autres modules matériels du système.

Un ISS est un modèle de simulation habituellement codé dans un langage de haut niveau qui émule le comportement d'un microprocesseur en lisant ses instructions d'un binaire exécutables spécifié au début de la simulation et en représentant l'état du microprocesseur dans des variables internes (pseudo-registres) [19]. L'ISS intègre quelques fois la simulation de quelques périphériques de base comme un *timer*, les interruptions, les ports d'entrée-sortie, etc. Le binaire exécutable de la simulation est chargé en mémoire utilisateur du processus de simulation, i.e. son code n'est jamais réellement exécuté par le processeur hôte. L'exécution d'un ISS se produit généralement dans les étapes suivantes :

1. Détermine l'adresse de la première instruction à charger (dans la zone mémoire contenant le binaire exécutable).
2. Charge l'instruction dans une variable interne.
3. Dépendamment de l'instruction chargée, l'ISS exécute une fonction émulant le comportement de ladite instruction. Les calculs et les accès mémoire pour récupérer les opérandes sont effectués ici.
4. Détermine l'adresse de la prochaine instruction à exécuter.
5. Reprend à l'étape 2.

Dans [20], les auteurs proposent une classification des ISS en 4 vues de modélisation :

- **Vue Fonctionnelle** pour développer et vérifier des algorithmes. Seulement les instructions sont modélisées et leur exécution séquentielle.
- **Vue Programmeur** pour le développement et la vérification logicielle. Les fonctionnalités visible au programmeur (cache, timer, DCR, etc.) sont implémentées. Cette méthode est environ 500 fois plus rapide qu'une simulation RTL[5].
- **Vue Architecturale** qui utilise un modèle approximativement précis au cycle. Elle est utilisée pour l'exploration et la vérification architecturale. Le pipeline du processeur est implémenté. Il est possible d'atteindre une précision sur les cycles de simulation supérieure à 95% et des simulations 20 fois plus rapide qu'avec un modèle RTL. Dans [21], une méthode de pré-évaluation des résultats intermédiaires des étages du pipeline est proposée pour réduire l'impact de la gestion d'événements concurrents sur la vitesse de simulation.
- **Vue Vérification** est précise au cycle près et est utile pour la vérification matérielle.

Selon [22], différentes méthodes peuvent être utilisées pour accélérer un ISS développé en SystemC :

- Minimiser la lecture sur des ports SystemC ;
- Combiner si possible en un seul *thread* les éléments concurrents
- Supprimer l'activité de mémoire d'instruction pour réduire les délais engendrés par le bus partagé.
- Intercepter certaines fonctions du kernel, tel *memset* et *memcpy*, et les exécuter nativement dans l'ISS.

---

[5] Résultats obtenus dans [20] pour des ISS codés en C pur et encapsulés dans des *wrappers* SystemC.

Le logiciel s'exécutant sur l'ISS peut interagir de différentes manières avec le reste du système. Dans [23], l'ISS modélise en C le processeur, le bus de communication et la mémoire en une seule entité. Pour faire communiquer ce modèle avec le reste de la simulation SystemC, l'accès à certaines plages d'adresse de la mémoire ne passe pas par la mémoire de la machine hôte mais par un adaptateur auquel est directement connecté le module destinataire de la transaction. Dans [24] et [25], les auteurs proposent une réutilisation du potentiel du débogueur GNU GDB qui inclut une multitude de simulateurs de processeur. Le processus se base sur l'utilisation de l'interface de débogage à distance de GDB (RDI) pour relier l'ISS à la simulation SystemC via un *wrapper*. Ce *wrapper* encapsule toutes les communications interprocessus (IPC) comme des pipes, des sockets, etc. Il est responsable de la synchronisation interprocessus et de la traduction des informations provenant de GDB en transaction bus. Cette méthode nécessite cependant des modifications au kernel de SystemC. Une dernière méthode [26] est proposée pour réduire les immenses délais occasionnés par des appels IPC fréquents. Elle réutilise le code source disponible. Il faut instancier l'ISS comme classe C++ à l'intérieur de la simulation SystemC. Il faut encapsuler l'ISS dans un wrapper et chaque fonctionnalité (boucle de décodage, accès mémoire en entrée, accès mémoire en sortie, etc.) est gérée par un thread SystemC différent qui communique à l'aide de signaux et d'événements de synchronisation.

ANNEXE D

SPACE : UNE PLATEFORME DE DÉVELOPPEMENT ESL

SPACE [5][27] est un environnement de développement utilisant la librairie SystemC et une méthodologie de codesign pour effectuer la modélisation, l'exploration architecturale et la validation de SoC au niveau système. SPACE est composé de deux parties distinctes, soit SpaceLib, une librairie de simulation incluant des modèles de composantes diverses (bus, ISS, adaptateurs de communication, timer, etc.), et SpaceStudio, un outil graphique facilitant le développement et la production d'une simulation par la prise en charge visuelle des paradigmes de modélisation et de communication SPACE. SPACE améliore une méthodologie de conception entièrement basée sur SystemC avec trois concepts importants :

1. **Support homogène de la partition logicielle** : SPACE permet la modélisation et la simulation du logiciel d'un MPSoC. La librairie SpaceLib intègre un RTOS (µC/OS-II) afin de permettre la conception de systèmes temps réel. Deux ISS étaient disponibles (µBlaze et ARM) avant le début de ce projet. De plus, les modules logiciels SPACE peuvent être déplacés en matériel et vice versa, de manière totalement transparente pour l'utilisateur et sans **aucune** modification au code.
2. **Réutilisation de blocs IP** : SPACE offre un ensemble de modèles de composants standard pouvant être réutilisés dans différentes architectures à bas niveau.
3. **Généricité du protocole de communication** : SPACE offre un protocole de communication basé sur la passation de messages ainsi que deux types de primitives de communication, soit bloquantes et non-bloquantes. Les primitives de communication utilisent les identificateurs de modules pour faire transiger les messages dans le réseau d'interconnexions qui est totalement abstrait au niveau

de la fonctionnalité des modules. La seule architecture présentement supportée est celle de CoreConnect d'IBM.

D.1. Les niveaux de raffinement architectural

SPACE offre actuellement deux niveaux de raffinement architectural :

- **Elix :** Ce niveau d'abstraction permet la validation de la fonctionnalité d'un système avec ou sans quelques notions temporelles de délais causés par un protocole de communication (paramétrable et configurable pour émuler un standard). Les opérations de communication se font par des appels aux fonctions des primitives de communication. Les messages transigent par un modèle abstrait de bus ne correspondant à aucun standard et servant simplement à effectuer un routage des messages. Au niveau Elix, aucun partitionnement logiciel/matériel n'est encore effectué. Elix correspond aux niveaux TF et UTF de SystemC.
- **Simtek :** À ce niveau d'abstraction est effectué le partitionnement logiciel/matériel des modules dont la fonctionnalité a été validée à Elix. Un protocole de communication spécifique est implémenté par un modèle de bus précis au cycle près. L'architecture ainsi obtenue est très semblable à celle au niveau RTL. Il est donc possible de valider le logiciel, le matériel ainsi que les interfaces de communications. Simtek permet aussi l'exploration architecturale en facilitant le déplacement des modules.

D.2. Concepts de base

D.2.1. Modules

Un module représente l'unité fonctionnelle de base d'une architecture SPACE et implémente une fonctionnalité précise. C'est une instance de la classe

SpaceBaseModule, elle-même dérivant d'un module SystemC. Au niveau Simtek, le module peut être logiciel ou matériel, indifféremment du code qu'il contient. Lorsqu'il est logiciel, un module est conceptualisé sous la forme d'une tâche logicielle s'exécutant sur un RTOS.

Chaque module possède sont identificateur numérique unique permettant d'abstraire les paramètres de source et de destination des communications. Dans une architecture de base, le module est connecté au média de communication par un adaptateur (*wrapper*) qui assure le découplage de l'interface module/bus et l'encapsulation de la conversion de protocole de communication.

De cette manière, les communications peuvent être effectuées indifféremment du niveau d'abstraction du bus ou de l'interface qu'il implémente. SpaceBaseModule a donc un port de communication unique (**SpaceModuleIF**) sur lequel se branche l'adaptateur approprié selon l'architecture de communication choisie.

D.2.1. Architecture

Une architecture SPACE contient des modules utilisateurs implantant la fonctionnalité d'un système, des composants standard de la librairie SpaceLib et un médium de communication. Une architecture peut être modélisée à Elix ou à Simtek.

Dans l'exemple d'architecture SPACE au niveau Simtek de la Figure , les composants ISS, ISSAdapter (adaptateur de communication matériel-logiciel), PIC, Timer et SDRAM sont des composants de SpaceLib. La mémoire *CodeRam* contient le binaire exécutable du logiciel simulé sur l'ISS (modèle de simulation du processeur). Les blocs étiquetés *IPIF & Space Wrapper* permettent la communication des modules utilisateurs avec le reste de la plateforme.

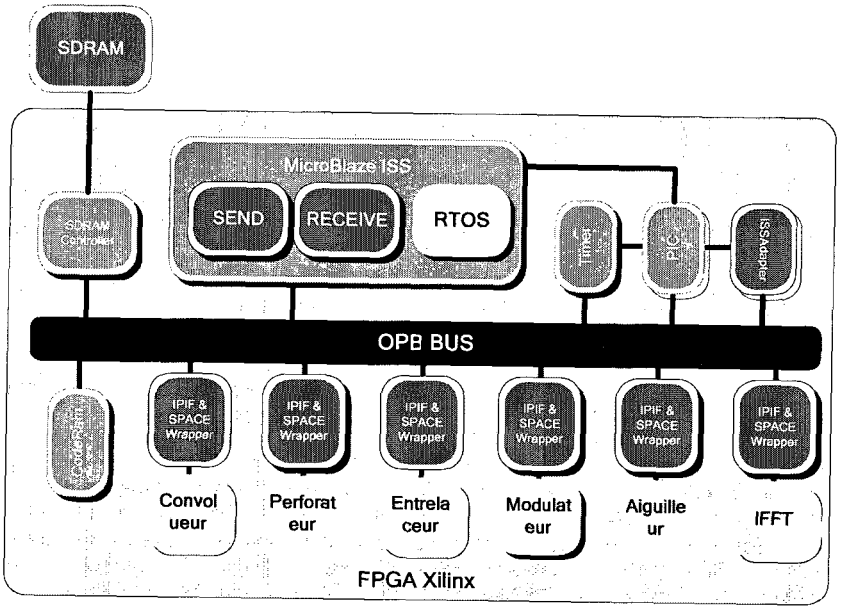


Figure D.1 – Exemple d'architecture SPACE à Simtek

ANNEXE E

LE PROCESSEUR : BLOC DE BASE DE LA CONCEPTION

Les microprocesseurs s'imposent de plus en plus comme éléments de base de la conception de systèmes embarqués [28], d'où l'émergence récente des MPSoC. Un tour d'horizon des avenues de design possibles s'impose donc.

5.8.1. Les processeurs *soft*

Les processeurs *soft* sont des microprocesseurs qui sont entièrement implémentés en utilisant une synthèse logique sur un circuit contenant de la logique programmable comme un FPGA ou un CPLD. Les exemples les plus connus sont le μBlaze de Xilinx et le NIOS-II d'Altera.

5.8.2. Les processeurs extensibles

Les plates-formes à processeurs extensibles sont devenues une possibilité pour surmonter la crise de complexité dans la conception des SoC. Les processeurs extensibles présentent généralement trois caractéristiques principales [29] permettant d'améliorer les performances de la partition logicielle:

- **Extensibilité des instructions :** Possibilité d'étendre les fonctionnalités du processeur en ajoutant des unités de calcul spécialisées et des registres à même le chemin de données en décrivant leur comportement fonctionnel dans un langage HDL.
- **Inclusion/Exclusion de blocs prédéfinis :** Le concepteur peut choisir d'ajouter ou retirer certains blocs du processeur tel des caches, des registres spéciaux, des blocs de multiplication-accumulation (MAC), un FPU, etc.

- **Paramétrisation :** Le concepteur peut fixer certains paramètres du processeur comme la taille des caches, le nombre de registres généraux, la largeur des interfaces bus, la taille des codes d'opération des instructions, etc.

Les exemples les plus connus sont le NIOS-II d'Altera, le Tensilica Xtensa [30][28] et le PowerPC405 FX de Xilinx.

ANNEXE F

LA PILE LOGICIELLE

La Figure F.1 illustre la structure stratifiée de la pile logicielle. Cette structure favorise la réutilisation et la modularité des composants logiciels. Elle permet aussi d'exécuter un même logiciel applicatif sur différentes plateformes en substituant les couches inférieures mais en offrant les mêmes services.

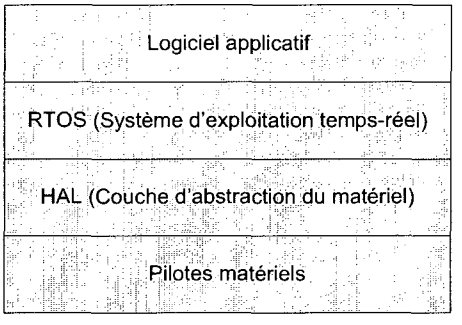


Figure F.1 – La structure de la pile logicielle

Le niveau supérieur implante le comportement du logiciel applicatif. C'est le cœur de l'application typiquement codé dans un langage de haut niveau.

La deuxième couche est facultative. Il est probable que le logiciel applicatif nécessite l'utilisation de certains mécanismes de temps réel d'où la présence d'un RTOS (« Real Time Operating System »). Gérant habituellement les ressources logicielles et matérielles dans un environnement multitâche – en fournissant des mécanismes comme les sémaphores, les mutex, etc. – le RTOS assure en plus le déterminisme de son exécution et le respect de contraintes de temps. Les exemples sont multiples : MicroC/OS-II [31] VxWorks, QNX, RTLinux, etc.

La troisième couche, soit la couche d'abstraction du matériel ou « Hardware Abstraction Layer » (HAL) en anglais, est elle aussi facultative. Elle sert d'interface entre les couches supérieures et les pilotes de périphérique en abstrayant les détails de bas niveau.

La quatrième couche, nécessaire, nommé la couche des pilotes (ou « drivers ») sert à interfacer le matériel du système. Chaque composant de la partition matérielle accessible au logiciel (e.g. contrôleur d'interruption, minuterie, UART, etc.) possède son propre pilote offrant des méthodes spécifiques à ses fonctionnalités.

ANNEXE G

LATENCES LOGICIELLES AVEC DIRECTLINK

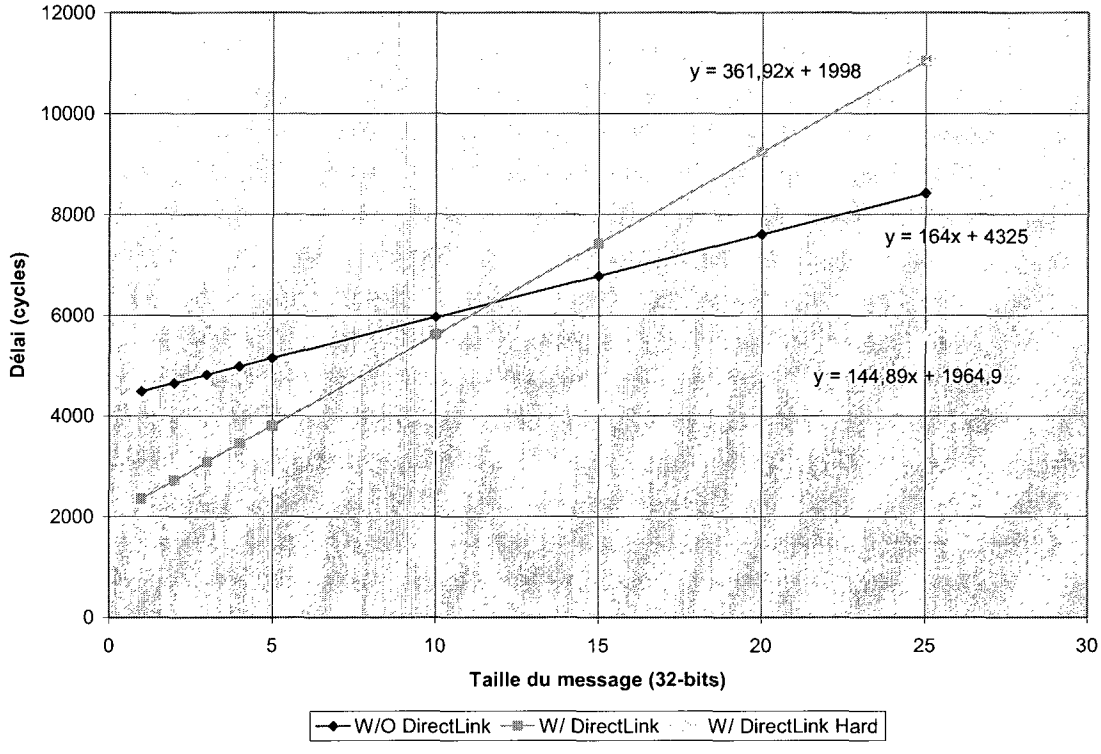


Figure G.1 – Délais d'écriture via DirectLink pour le µBlaze

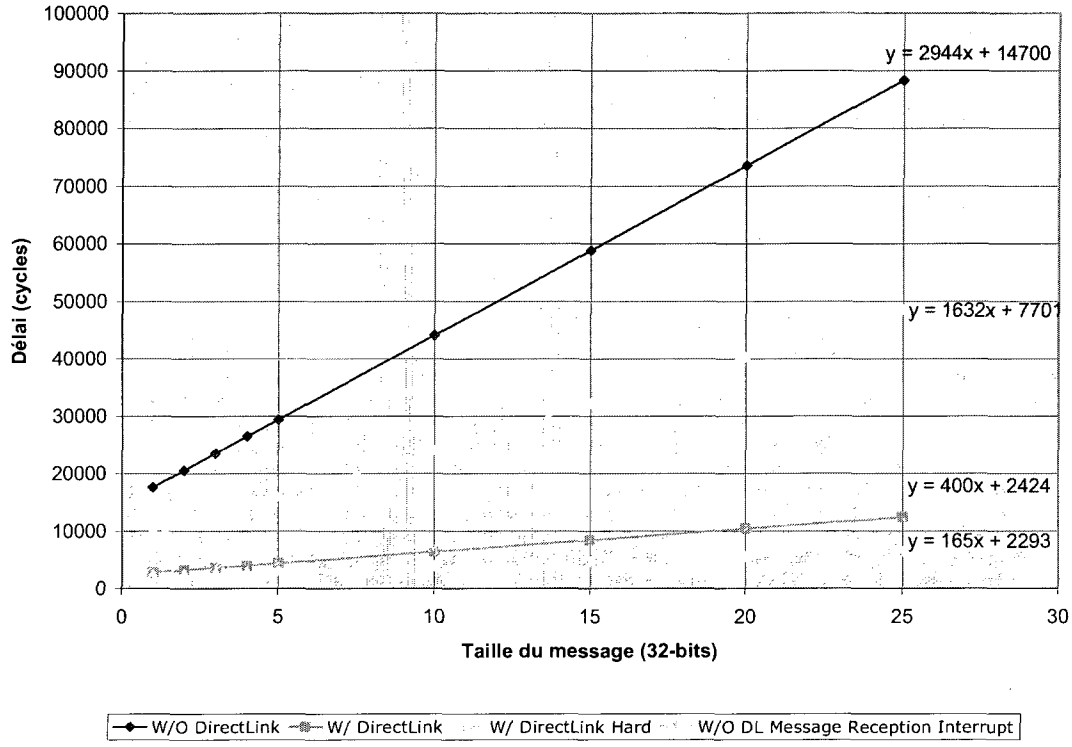


Figure G.2 – Délais de lecture via DirectLink pour le µBlaze

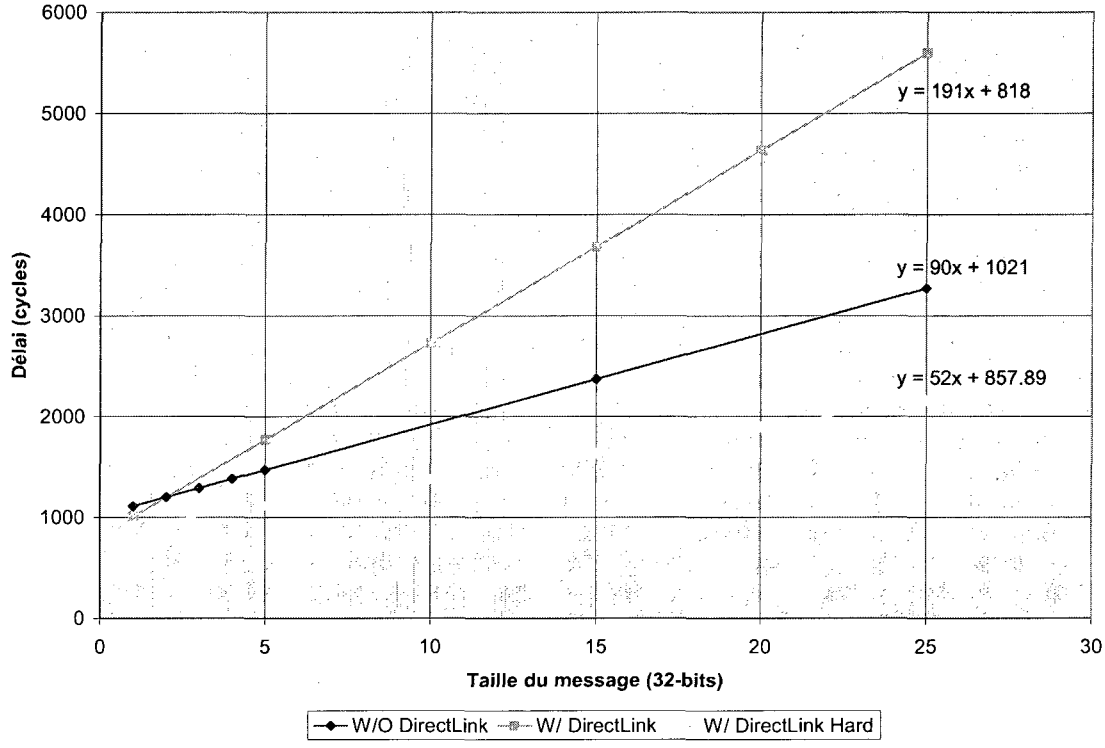


Figure G.3 – Délais d'écriture via DirectLink pour le PowerPC405

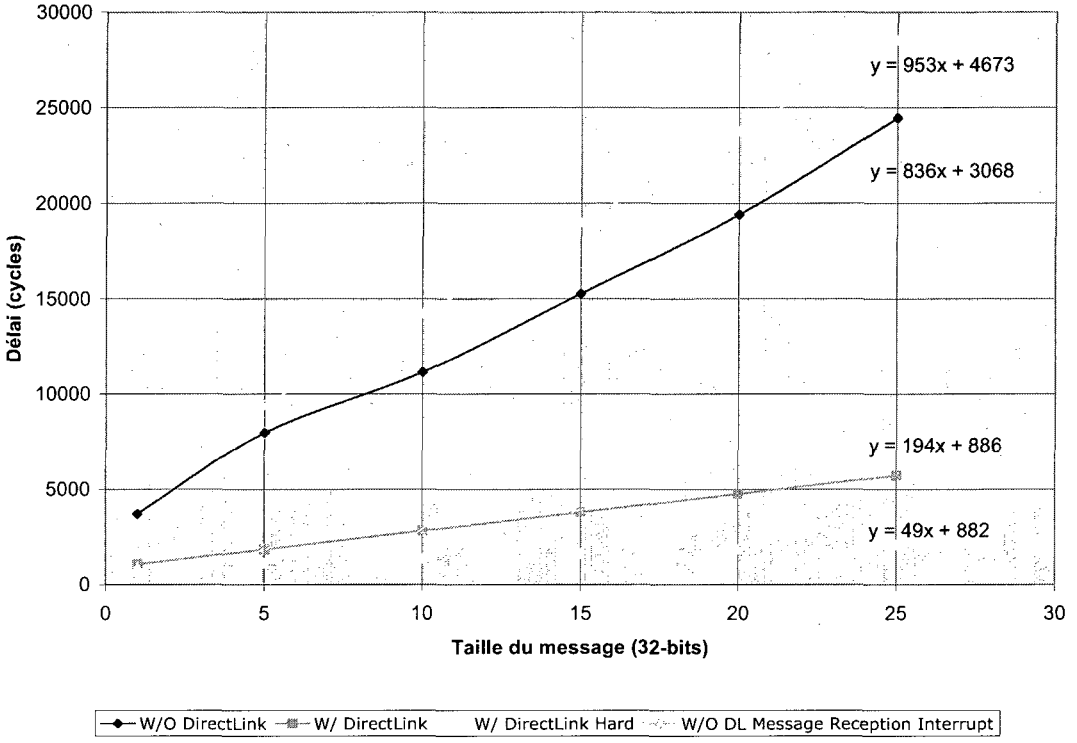


Figure G.4 – Délais de lecture via DirectLink pour le PowerPC405