THÈSE D'HABILITATION DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ : INFORMATIQUE

Présentée par Frédéric PÉTROT
pour l'obtention d'une
habilitation à diriger des recherches
sur le thème :

INTÉGRATION DES SYSTÈMES MATÉRIEL/LOGICIEL

À soutenir le 18 juin 2003 devant le jury composé de

Jean-Paul Calvez
Paul Feautrier
Alain Greiner
Ahmed Jerraya
Luciano Lavagno
Michel Robert
Yves Sorel

Table des matières

1 Introduction **17**
1.1 Bibliographie 20
1.2 Publications relatives au chapitre 1 20

I CAO de bas niveau et micro-architecture **21**

2 Plate-forme de conception pour circuits intégrés VLSI **23**
2.1 Introduction 23
2.2 Développement de l'infrastructure d'**Alliance** 24
2.3 Sur l'encodage des états des machines à état 25
2.4 Analyse sémantique des descriptions VHDL pour la synthèse 27
2.5 Plot d'entrée/sortie trois états pour bus PCI 29
2.6 Apports d'**Alliance** 29
2.7 Prototypes logiciels 30
2.8 Bibliographie 30
2.9 Publications relatives au chapitre 2 31
2.10 Participation à l'encadrement de thèses 32

II Systèmes intégrés **33**

3 Introduction **35**
3.1 Problématique générale 35
3.2 Méthode 40
3.3 Bibliographie 43
3.4 Publications relatives au chapitre 3 43

4 Spécification fonctionnelle parallèle exécutable **45**
4.1 Introduction 45
4.2 Problématique 45
4.3 Primitives de communication 45
4.4 Raffinements 48
4.5 Extensions et perspectives 49
4.6 Prototypes logiciels 50
4.7 Bibliographie 50
4.8 Publications relatives au chapitre 4 51
4.9 Encadrement de thèses 51
4.10 Encadrement de stages de DEA 51

5 Simulation « au niveau cycle » de systèmes digitaux synchrones 53
5.1 Introduction . . . 53
5.2 Problématique . . . 53
5.3 Approche retenue . . . 54
5.4 Évaluation d'un système . . . 55
5.5 Simulation de coprocesseurs matériels avant synthèse . . . 58
5.6 Simulation de modèles VHDL . . . 58
5.7 Performances de CASS et comparaison avec SystemC . . . 59
5.8 Évaluation de la consommation au niveau cycle . . . 60
5.9 Conclusion et perspectives . . . 60
5.10 Prototypes logiciels . . . 61
5.11 Collaborations avec l'industrie . . . 61
5.12 Bibliographie . . . 61
5.13 Publications relatives au chapitre 5 . . . 62
5.14 Encadrement de stage de DEA . . . 63
5.15 Encadrement de thèses . . . 63

6 Synthèse des communications et support matériel/logiciel 65
6.1 Introduction . . . 65
6.2 Approche générale . . . 66
6.3 Module matériel de support à la communication . . . 67
6.4 Prototypes logiciels et matériels . . . 70
6.5 Collaborations avec l'industrie . . . 70
6.6 Bibliographie . . . 71
6.7 Publications relatives au chapitre 6 . . . 71
6.8 Encadrement de thèses . . . 71
6.9 Encadrement de stages de DEA . . . 71

7 Synthèse d'architecture 73
7.1 Introduction . . . 73
7.2 Approche générale . . . 74
7.3 Exemple d'utilisation . . . 76
7.4 Conclusion et perspectives . . . 77
7.5 Prototypes logiciels . . . 77
7.6 Collaborations avec l'industrie . . . 78
7.7 Bibliographie . . . 78
7.8 Publications relatives au chapitre 7 . . . 78
7.9 Encadrement de thèses . . . 79
7.10 Encadrement de stages de DEA . . . 79

8 Noyau de système d'exploitation distribué embarqué 81
8.1 Introduction . . . 81
8.2 Problématique . . . 81
8.3 Approche retenue . . . 82
8.4 Implantation . . . 85
8.5 Autres problèmes liés à l'architecture . . . 87
8.6 Performances . . . 88
8.7 Conclusion et perspectives . . . 89

8.8 Bibliographie 89
8.9 Publications relatives au chapitre 8 90
8.10 Encadrement de thèse 90
8.11 Encadrement de stages de DEA 90
8.12 Prototypes 90

9 Prospective **91**

10 Conclusion **93**

Table des figures

1.1 Évolution de la longueur minimal du canal du transistor. 17
1.2 Évolution de la surface maximum des circuits. 18
1.3 Évolution du diamètre de la galette de silicium (*wafer*). 18
1.4 Évolution du nombre de portes élémentaires entre deux registres. 19
1.5 Évolution de la longueur des pistes d'interconnexion. 19

2.1 Architecture logicielle d'**Alliance** 24
2.2 Rapport des activités théoriques 26

3.1 Une application décrite comme un graphe de tâches communicantes. 36
3.2 Illustration de la loi d'Amdahl 36
3.3 Choix de l'architecture de communication du système. 38
3.4 Affectation des tâches. 39
3.5 Scénario de conception de systèmes intégrés 41

4.1 Un canal de communication entre deux tâches 47

5.1 Décomposition d'un automate pour la simulation cycle 56
5.2 Boucle de simulation d'un système composé uniquement de machines de Moore 56
5.3 Construction d'un graphe acyclique 57
5.4 Construction d'un graphe avec un cycle 57
5.5 *threads* exécutées 58

6.1 Affectation des tâches et des modes de communications 65
6.2 Écriture dans un canal en utilisant les services des *threads* POSIX 67
6.3 Lecture dans un canal en utilisant les services des *threads* POSIX 68
6.4 Niveaux d'abstraction visibles lors de l'implantation. 69
6.5 Vue d'ensemble du module d'interface. 69
6.6 Vision VCI d'une plate-forme. 70

7.1 Présentation générale de la méthode Ugh 74
7.2 Connexion d'un coprocesseur à un bus. 75
7.3 Environnement de validation du VLD. 76
7.4 Temps d'exécution du VLD sur une séquence MPEG2. 77

8.1 États possibles d'un *thread*. 86
8.2 Structure de l'ordonnanceur. 86

Liste des tableaux

5.1 Comparaison des performances de CASS et de SystemC. 60

8.1 Mesure de performance, en nombre de cycles, de quelques fonctions du noyau. . 88

Résumé

Ce document retrace mes activités de recherche depuis ma soutenance de thèse en 1994. Y sont présentés des travaux achevés, ainsi que des travaux en cours et d'autres encore dans un stade exploratoire.

Mes travaux ont porté sur deux sujets assez différents. De 1994 à 1999, j'ai continué à m'intéresser aux aspects bas niveaux de la CAO pour l'électronique numérique, dans la suite logique de mon travail de thèse. Ces travaux ont contribués à etoffer la chaîne de CAO **Alliance**, dont je suis l'un des principaux architectes et contributeurs.

Depuis 1996, je me suis intéréssé aux aspects de l'intégration des systèmes numériques sur une puce. Ces systèmes ont la particularité d'être dédiés à une application, et de ne pas être disponible matériellement, à la différence des systèmes embarqués classiques. Le document détaille la problèmatique des *Systems on Chip*, et présente une méthode de conception systématique pour les applications modélisables sous forme de graphe de tâches qui communiquent par des files d'attentes sans pertes à implanter sur des architectures multiprocesseur à mémoire partagée. La mise en œuvre de cette méthode, non entièrement automatique, est possible grâce à 1) une modélisation exécutable de l'application, 2) une modélisation simulable de l'architecture cible, 3) des outils permettant la projection de l'application sur l'architecture cible. Des prototypes permettant l'utilisation de cette méthode ont été définis et développés, et sont disponibles dans l'environnement **Disydent**.

Les travaux en cours portent sur l'extension des différents prototypes, comme par exemple permettre l'utilisation d'une primitive de communication supplémentaire lors de la specification. Ceci remet en cause les propriétés initiales du modèle, mais s'avère nécessaire pour bon nombre d'applications.

Les travaux futurs portent principalement sur deux axes. Le premier concerne l'évaluation de performances avant implantation, ce qui permet une simulation très rapide. La contrepartie est une perte de précision importante car l'architecture n'est décrite que grossièrement. Les problèmes essentiels étant liés à la contention, et donc aux dates d'arrivée des données, toute la question est de réussir à obtenir des estimations réalistes à ce niveau. Le second est du au fait que beaucoup d'applications intégrées ont des contraintes temps réel. Hélas, la caractérisation de leur comportement temporel sur des architectures multiprocesseur est loin d'être simple, et aussi bien la le calcul que la prise en compte de ces contraintes est en soit une gageure.

Abstract

This document traces my research activities since the defense of my Ph. D in 1994. Achieved, current and future works are presented here.

This work is related to two different subjects. From 1994 to 1999, I have been involved in back-end CAD tools for VLSI design, as a follow-up of my thesis. These tools have been added to the **Alliance** CAD System, whom I am one of the main architects and contributors.

Since 1996, I've worked on application integration and digital SoC design. Those systems are usually not available as PCB, unlike classical embedded systems. The document details the problem at hand, and proposes a systematic method of integration for applications modeled as tasks communicating through lossless fifos to be mapped on shared memory multiprocessor architectures. The implementation of this non fully automated approach is possible thanks to 1° an executable model of the application, 2° a simulable model of the architecture, 3° tools that allow to map the application on the target architecture. Prototypes tools have been developed and are available in the **Disydent** environment.

Current works tend to extend the capabilities of the tools. As an example, a new communication primitive is necessary to handle more complexe communication schemes, but its introduction does not preserve the initial properties of the model.

Future works are mainly starting in two directions. Firstly, high level performance estimation of the application for a given architecture. This allows fast simulation, but loss of precision. Since communication contention must be modeled, the whole question is to be able to have fairly accurate delays on data exchanges. Secondly, most embedded applications have real time constraints. Unfortunately, the temporal caracterization of such applications on multiprocessor architecture is not simple at all, and both the computation and the handling of these constraints are great challenges.

Prologue

Ce document présente les axes de recherche auxquels je me suis intéressés depuis une dizaine d'année à l'Université Pierre et Marie Curie. Ces travaux se sont déroulés en pratique au sein de la même équipe, précédemment nommée CAO et VLSI du Laboratoire Méthodologie et Architecture des Systèmes Informatiques de l'Institut Blaise Pascal, rebaptisée équipe Architecture des Systèmes Intégrés et Micro-électronique (ASIM) du Laboratoire d'Informatique de Paris VI (LIP6) en 1996.

Ces travaux couvrent un large spectre, qui va de la participation à la conception et à l'implémentation d'une plateforme pour le développement d'outils de CAO pour circuits intégrés, ce qui fut l'objet de ma thèse soutenue en 1994, à la definition de l'architecture de modèles de communication dans les systèmes intégrés en passant par la synthèse de haut niveau et les micro-noyaux logiciels, qui est actuellement une partie importante de mes recherches.

Cette grande fourchette d'intérêts peut-être préssentie comme étant trop vaste dans le sens où, comme le dit l'adage, « qui trop embrasse mal étreint ». Elle peut également être vue comme légitime, car elle permet de comprendre le vaste domaine de la conception de circuits et de systèmes globalement, en faisant le liens entre les différents niveaux de représentation, et en connaissant à chaque fois leurs atouts et leurs limites.

Ces travaux ont nécessité beaucoup d'ingéniérie pour aboutir à des résultats expérimentaux mesurables, qui se traduisent par des prototypes logiciels. Ils ont quasiment tous pris place, avec différents niveaux d'implication, dans des projets européens (Esprit, Medea) en collaboration avec les principaux acteurs industriels européens du secteur.

Chapitre 1

Introduction

En prédisant en 1965[1] que la densité d'intégration va grosso-modo doubler tous les 18 mois[Moo65], Gordon Moore définit en fait une ligne de conduite pour l'industrie du semi-conducteur [Sch99] qui reste encore valable aujourd'hui. La meilleure illustration de cette « loi de Moore » est fournie par deux grandeurs technologiques : la taille minimum de la longueur électrique du canal du transistor – figure 1.1 – et la densité de défaut, qui défini en pratique la taille maximale de la surface des circuits intégrés – figure 1.2 –. Le premier facteur permet d'in-

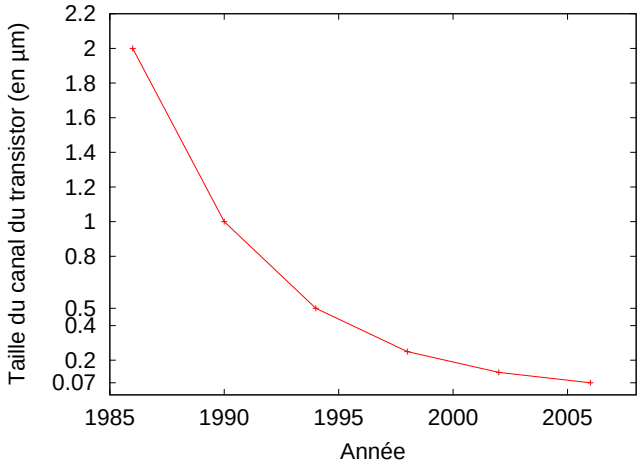

FIG. 1.1 – Évolution de la longueur minimal du canal du transistor.

tégrer plus de transistors sur la même surface et le second permet d'agrandir la surface. Leur conjonction permet donc de suivre la courbe de la « loi de Moore ».

Ces évolutions sont très coûteuses pour les vendeurs de silicium, mais ces investissement peuvent être fort rentables. En effet, le marché du semi-conducteur, bien que fortement instable, croît en moyenne de 14% par an depuis 1960[Hut96]. Prenons l'exemple du diamètre de la galette de silicium (*wafer*) – figure 1.3 –. Son accroissement permet de réaliser une plus grande quantité de circuits en même temps, donc d'améliorer les coûts de productions. C'est un des facteur qui fait que, dans le même temps, le prix des appareils électroniques a fortement baissé. Pour illustrer cette baisse, prenons le prix, emblématique, du bit de mémoire dynamique. Il a baissé de 33% par an depuis l'énoncé de la « loi de Moore » : en 1970, il était de 2 cents et, en

[1] À ce moment là, le circuit intégré le plus complexe est constitué de 64 composants.

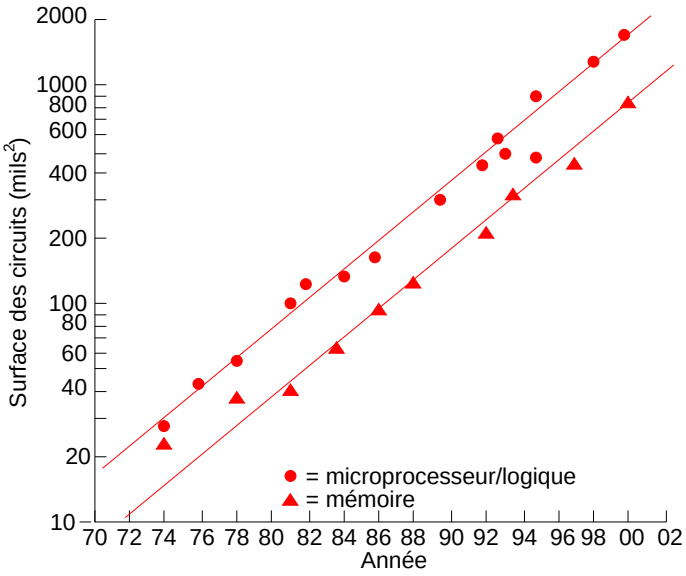

FIG. 1.2 – Évolution de la surface maximum des circuits.

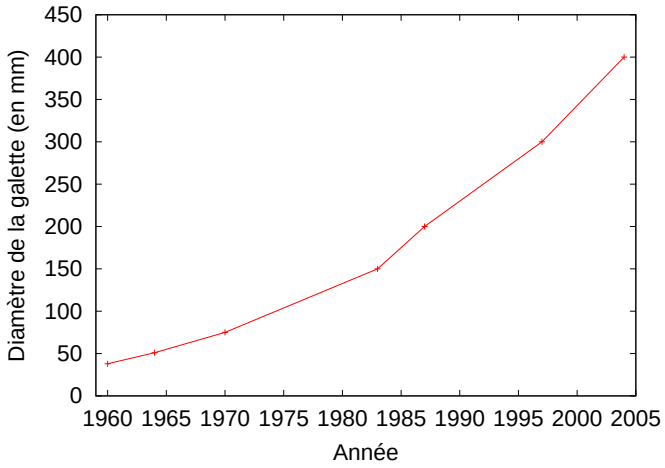

FIG. 1.3 – Évolution du diamètre de la galette de silicium (*wafer*).

2000, il est de l'ordre de 10^{-5} cents.

Comme toute loi exponentielle, la « loi de Moore » possède ses limites dans sa formulation actuelle. D'une part, la physique nous enseigne qu'on ne peut pas indéfiniment réduire la taille d'un canal de transistor. Cette limite est intrinsèque, et les prouesses technologiques pourront au plus permettre de s'en rapprocher. Ce point est laissé aux physiciens et technologues. D'autre part, le coût des usines de fabrication devient exorbitant, et seuls des consortiums constitués des « grands » du secteur peuvent réaliser les investissements nécessaires. Ce point est laissé aux économistes et aux financiers. Finalement, et c'est là que se situe notre intérêt, la complexité croissante des circuits fait que leur conception, c.-à-d. rendre utiles les centaines de millions de transistors dont on dispose, est de plus en plus difficile. Pour tirer un meilleur parti du parallélisme intrinsèque du matériel, les concepteurs ont profondément modifié leur manière de voir l'architecture des circuits. La figure 1.4 illustre l'évolution du nombre de portes élémentaires entre deux registres dans les circuits intégrés spécifiques à une application (ASIC). Cela illustre très clairement que la recherche de la performance n'est pas qu'une question de technologie,

mais aussi, et grandement, une affaire de conception.

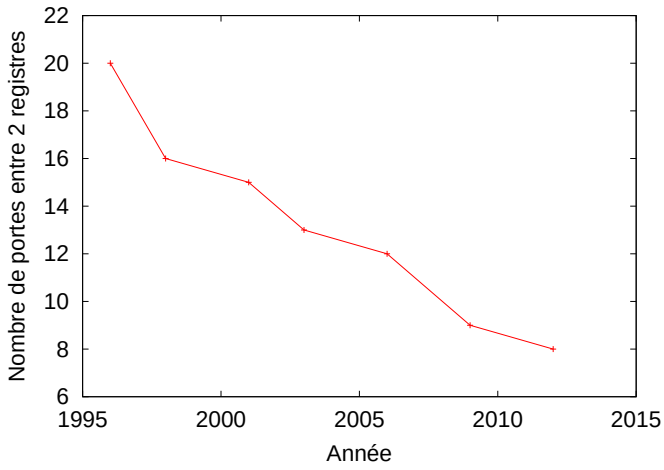

FIG. 1.4 – Évolution du nombre de portes élémentaires entre deux registres.

Cette complexité ne peut être gérée que grâce à l'automatisation de la quasi-totalité des étapes de la conception, d'autant plus que la durée de vie des produits sur le marché est assez réduite, et que donc les ingénieurs sont soumis à des contraintes de temps de conception très ferme. La complexité d'un circuit peut grossièrement être ramené au nombre de transistors qui le composent. Une autre grandeur illustrant cette complexité est la longueur cumulée des interconnexions sur les circuits intégrés [Isa00]. La figure 1.5 illustre son évolution. De telles

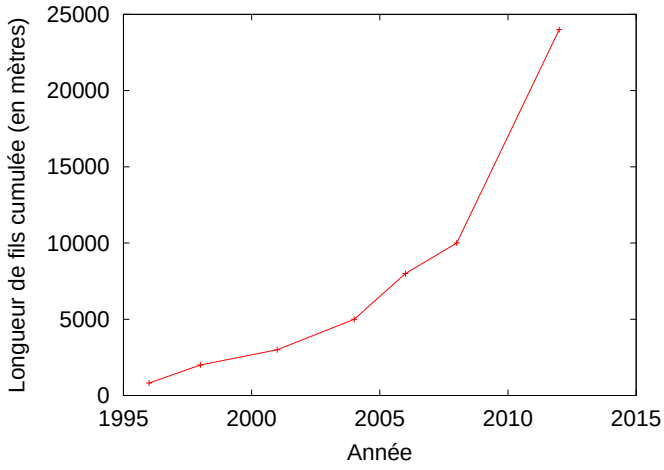

FIG. 1.5 – Évolution de la longueur des pistes d'interconnexion.

complexités exigent des méthodes de conception très bien définies et des outils de conception assistée par ordinateur extrêmement puissants. Il faut également remarquer que ces méthodes doivent favoriser la réutilisation de l'existant, et que le nombre de transistors nouveaux entre deux générations d'un même produit tend à diminuer fortement.

Les travaux que j'ai mené et que je continue à mener visent précisément à rendre appréhendable cette complexité. Dans ce rapport, je décris les deux grands axes de travail auxquels je me suis intéressé.

J'ai participé à la définition et à l'implantation de la chaîne de CAO **Alliance** qui s'intéresse à ces aspects bas-niveau de la conception. Je décris brièvement le travail fait dans ce cadre durant

la première partie du manuscrit, en insistant particulièrement sur les points nouveau depuis la soutenance de ma thèse.

Si regarder l'évolution des semi-conducteurs par ces grandeurs met en exergue la complexité combinatoire intrinsèque de la conception de circuit, elle ne met pas en évidence la complexité algorithmique des applications implantées, ni le besoin de méthodes plus abstraites facilitant la réutilisation.

Concevoir de nouveaux circuits aujourd'hui nécessite la réutilisation systématique de blocs déjà conçus, possiblement à différents niveaux de représentation. La deuxième partie du manuscrit s'intéresse à différents aspects de la conception de systèmes intégrés : spécification, synthèse des communication, simulation rapide, etc. J'y présente en détail les problématiques et des ébauches de solutions que l'équipe que je dirige a définie et implantée.

1.1 BIBLIOGRAPHIE

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. In *Electronics*, Volume 38, Number 8, pp. 114–117, April 1965.

[Hut96] G. Dan Hutcheson and Jerry D. Hutcheson. in *Scientific American*, Volume 274, Number 6, Technology and Economics in the Semiconductor Industry, January 1996, pp. 54–62

[Sch99] Robert R. Schaller. Technology Roadmaps : Implications for Innovation, Strategy and Policy. George Mason University, Fairfax, VA. PHD Dissertation, March 1999.

[Isa00] Randall B. Isaac. The Future of CMOS Technology. In *IBM Journal of Research and Development*, Volume 44, Number 3, pp. 369-378, May 2000.

1.2 PUBLICATIONS RELATIVES AU CHAPITRE 1

Contribution à ouvrages

[PW03] Frédéric Pétrot et Franck Wajsbürt. Article *circuits intégrés*. In *Encyclopædia Universalis*, version cédérom à paraître en 2003.

Première partie

CAO de bas niveau et micro-architecture

Chapitre 2

Plate-forme de conception pour circuits intégrés VLSI

2.1 INTRODUCTION

Dans ce chapitre, j'évoque les travaux que j'ai effectué autour de la plate-forme de CAO **Alliance** développé au LIP6.

Bref historique

Pour mémoire, l'idée de développer **Alliance**, une chaîne de CAO de circuits VLSI pour l'enseignement, est due à Gérard Noguez, qui avait également supervisé un début d'implantation jusqu'à sa disparition en 1989.

Sous l'impulsion d'Alain Greiner, cette idée a été reprise et étendue, avec l'ambition de faire d'**Alliance** également une plate-forme de développement permettant aux chercheurs en CAO de développer plus simplement les outils les plus avancés. En ce sens, le but d'**Alliance** est fondamentalement de fournir une infrastructure permettant de représenter les différentes vues d'un circuit à différents niveaux d'abstraction. Les principaux contributeurs, pour ce qui concerne la définition et l'implantation de cette infrastructure, sont : Alain Greiner, qui en a défini les grandes lignes et chapeauté la mise en œuvre, Pirouz Bazargan-Sabet, Luc Burgun, Ludovic Jacomme, Franck Wajsbürt et moi-même. L'originalité de l'équipe de recherche qui a développé **Alliance** est d'être composé également d'architectes et de concepteurs de circuits, à même de participer à la définition des outils et d'expérimenter les prototypes.

Sur cette infrastructure ont été bâtis environ 50 outils permettant de réaliser des circuits VLSI des spécifications au niveau transfert entre registres jusqu'au masques de fabrication.

J'ai eu la responsabilité d'une grande partie du développement de l'infrastructure d'**Alliance**, ainsi que de sa distribution et de son support de la fin de ma thèse en 1994 jusqu'en 1996. À cette date, environ 250 universités dans le monde avaient expérimenté **Alliance**, notamment l'Université des Nations-Unis qui l'avait choisit pour l'enseigement.

Travaux actuels

Depuis la fin de ma thèse, en juillet 1994, j'ai continué à m'intéresser à différents problèmes de la CAO des circuits *VLSI*. Afin de souligner la diversité des domaines étudiés, je détaille ci-dessous trois aspects sur lesquels j'ai plus précisément été amené à travailler. La première partie

concerne l'étude pratique de l'encodage des états d'une machine à états. La seconde concerne l'analyse sémantique des descriptions VHDL en vue de la synthèse au niveau transfert de registres. La dernière est orientée circuiterie, avec la définition et l'implantation d'un plot d'entrée/sortie pour le bus PCI.

J'ai mené également d'autres travaux, en collaboration avec différents chercheurs du département ASIM du LIP6, qui ont donné lieu à des publications dont la liste exhaustive, depuis la soutenance de ma thèse en juillet 1994, est donnée dans la section 2.9.

2.2 DÉVELOPPEMENT DE L'INFRASTRUCTURE D'**ALLIANCE**

L'architecture logicielle d'**Alliance** repose sur des structures de données partagées permettant de décrire chaque vue et niveau d'abstraction d'un circuit avec une sémantique bien définie. La figure 2.1 présente les principaux outils d'**Alliance** et leurs liens avec ces structures.

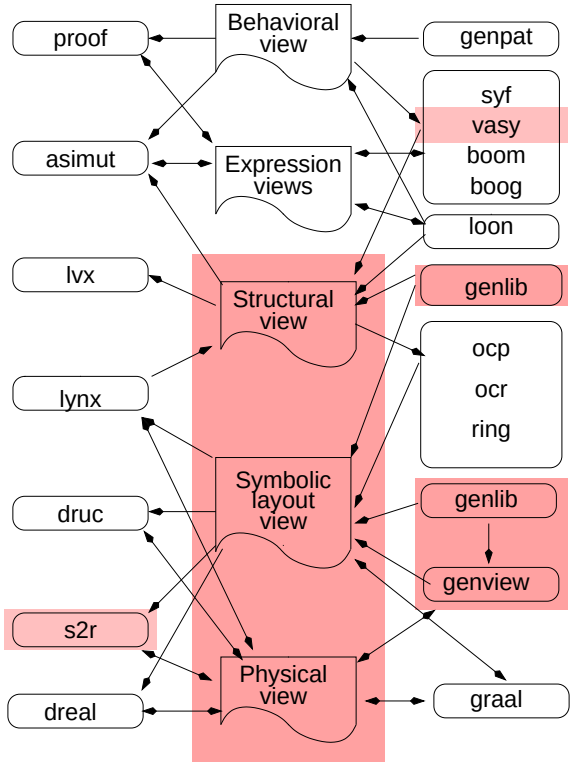

FIG. 2.1 – Architecture logicielle d'**Alliance**

J'ai été l'un des principaux contributeurs à la fois de la définition et du développement des structures de données permettant de représenter la vue structurelle – avec Alain Greiner et Pirouz Bazargan-Sabet –, de la vue physique symbolique – avec Alain Greiner et Franck Wajsbürt –, et de la vue physique réelle – avec Alain Greiner, Ludovic Jacomme et Franck Wajsbürt –. Je suis également le principal auteur des outils ou un contributeur significatif des outils grisés sur cette figure.

Le développement de ces infrastructures ainsi que des analyseurs permettant de lire et d'écrire les formats de fichiers les plus répandus pour chaque vue a demandé beaucoup d'ingénierie, pour aboutir à une interface de programmation (*API*) implantée et documentée correctement.

J'ai participé activement à ce développement et supervisé, de 1990 à 1994, de nombreux stages de maîtrise pour en réaliser l'implantation.

2.3 SUR L'ENCODAGE DES ÉTATS DES MACHINES À ÉTAT

Je me suis intéressé, avec Ludovic Jacomme et Julien Dunoyer, à l'étude de la consommation des automates d'état en fonction de l'encodage des états, pour des encodages d'entrée et de sortie donnés. C'est le cas général des machines à états fournies par un concepteur de circuits au niveau transfert de registres, pour lequel les communications entre un chemin de données et sa machine à états fournies au bit près font partie de la définition de la micro-architecture. Cette configuration est la configuration classique lorsque l'on utilise des opérateurs existants pour construire un chemin de données.

Le résultat principal de ce travail a été d'établir, au niveau macroscopique, des bornes probabilistes supérieures et inférieures décrivant l'activité des registres pour des machines encodées soit sur $\lceil \log_2 n \rceil$ bits, soit avec n bits pour n états (*one-hot*), ou n est le cardinal de l'ensemble des états. En faisant l'hypothèse, simplificatrice mais communément retenue, que la consommation d'un circuit est proportionnelle à l'activité de ses registres, on ramène le problème d'évaluation de la consommation au problème du calcul de cette activité. De même, on a supposé que la probabilité d'atteindre l'état s_i à partir de l'état s_j est de $\frac{1}{n}$ (hypothèse d'équiprobabilité). Cette hypothèse est grossièrement inexacte, mais elle permet de mener une étude théorique simple, ce qui n'est plus le cas si l'on possède des probabilités de transitions sur les arcs sortant de chaque état.

Dans la suite on note α_x l'activité de l'expérimentation x, et r_i le bit i du registre d'état r. Nous détaillons trois bornes calculées.

Borne pour un encodage aléatoire uniforme sur $\lceil \log_2 n \rceil$ bits Pour ce cas, nous examinons la probabilité qu'un bit donné du code ait la valeur r_j ou $\overline{r_j}$. Nous avons, par l'hypothèse d'uniformité, $\text{Prob}(r_j = 1) = \text{Prob}(r_j = 0) = \frac{1}{2}$, car tous les états ont la même probabilité d'être atteint. Il suffit alors de sommer ces probabilités sur la longueur du code pour obtenir l'activité moyenne du registre d'état.

$$\alpha_{ur} = \sum_{i=1}^{\lceil \log_2 n \rceil} \frac{1}{2} = \frac{\lceil \log_2 n \rceil}{2} \tag{2.1}$$

Borne inférieure sur $\lceil \log_2 n \rceil$ bits L'activité minimum est atteinte lorsque l'on a un codage des états qui suit un code Gray, et ce pour tous les chemins possibles dans l'automate. Chaque changement d'état provoque une unique transition sur la totalité du registre d'état. Ainsi, pour l'état S_j, nous avons les probabilités suivante pour chaque bit : $\text{Prob}(r_j \rightarrowtail \overline{r_j}) = 1$ pour $\lceil \log_2 n \rceil$ états sur $\lceil \log_2 n \rceil + 1$, et $\text{Prob}(r_j \rightarrowtail r_j) = 0$ pour 1 état sur $\lceil \log_2 n \rceil + 1$. En sommant sur tous les états pour lesquels la probabilité est non nulle, nous avons :

$$\alpha_{lb} = \sum_{i=1}^{\lceil \log_2 n \rceil} \frac{1}{\lceil \log_2 n \rceil + 1} = \frac{\lceil \log_2 n \rceil}{\lceil \log_2 n \rceil + 1} \tag{2.2}$$

Estimation pour un registre par état Là il ne s'agit plus de bornes, pour la raison que pour toutes les transitions hormis celle de l'état vers lui même, nous aurons deux inversions de bits dans le registre d'état. La probabilité qu'un état reboucle sur lui même, en considérant des états

équiprobables est : Prob$(S_i \rightarrow S_i) = \frac{1}{n}$, ainsi

$$\alpha_o = 2(1 - \frac{1}{n}) = 2 \times \frac{n-1}{n} \tag{2.3}$$

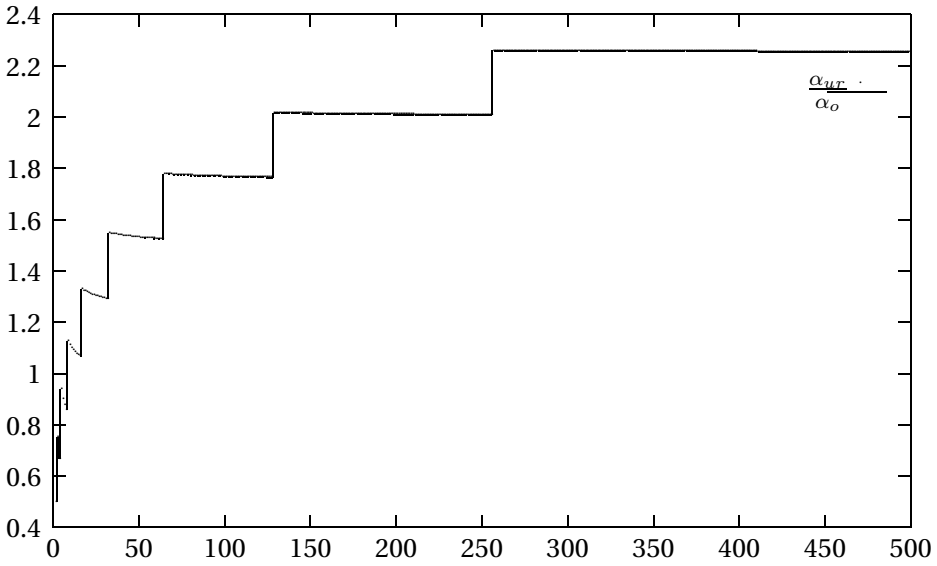

FIG. 2.2 – Rapport des activités théoriques

Comparaison théorique des activités La figure 2.2 trace $\frac{\alpha_{ur}}{\alpha_o}$, le gain de l'activité due à un encodage *one-hot* sur l'activité d'un encodage en $\lceil \log_2 n \rceil$ avec un comportement aléatoire uniforme, que nous pouvons comparer à ≈ 2 qui est le gain d'activité d'un encodage Gray sur l'activité d'un *one-hot*. Notons que le gain $\frac{\alpha_{ur}}{\alpha_o}$ devient supérieur à 1 pour $n = 9$.

Les algorithmes visant la minimisation de la consommation cherchent à encoder les chemins les plus parcourus par des séquences de type Gray. Pour ces chemins, la consommation minimale est atteinte. Pour les autres chemin, on peut supposer qu'ils seront encodés plus ou moins aléatoirement, et donc leur consommation excédera largement celle d'un encodage *one-hot*.

La figure 2.2 montre que pour des grosses machines à états, la consommation épargnée par les transitions de type Gray est moins importante que le coût d'une transition aléatoire. Malheureusement, et indépendamment de la qualité des heuristiques d'encodage, le nombre de machines à états encodable avec un code Gray est très petit. Ces machines ont des caractéristiques très spécifiques : un nœud ne peut pas avoir plus de $\lceil \log_2 n \rceil$ voisins ; si S_j possède une transition vers S_i alors S_i possède également une transition vers S_j ; deux états S_j et S_k, voisins de S_i, ne peuvent être voisins. Ainsi, pour les automates complexes, le rapport du nombre de transitions aléatoire sur le nombre de transitions Gray va croissant.

Cette étude théorique ne porte que sur l'activité des registres. Or pour confirmer ou infirmer l'intérêt de cette étude, il est nécessaire de mesurer la consommation en prenant en compte les parties combinatoires et les capacités des grilles et du routage. Une étude quantitative a été menée sur une centaine d'automates issus des bancs de test de Berkeley. Ces automates ont été encodés par diverses méthodes, synthétisés par Synopsys, puis placés, routés, et finalement extraits. Le signal de remise à zéro a été identifié afin de ne l'exciter qu'une unique fois, puis des vecteurs probabilistes ont été appliqués. De cette étude, il ressort que l'encodage *one-hot* produit des circuits conciliant une moindre consommation avec une surface plus petite et une

fréquence de fonctionnement plus élevée dès que le nombre d'états est de l'ordre de la dizaine. Ces résultats expérimentaux sont conformes à la théorie, et ce malgré des hypothèses de travail grossièrement simplificatrices.

Ces travaux ont été publiés dans deux ateliers, l'un national [DPJ97c] et l'autre international [DPJ97a], et dans deux conférences, l'une nationale [JP96] et l'autre internationale [DPJ97b].

2.4 ANALYSE SÉMANTIQUE DES DESCRIPTIONS VHDL POUR LA SYNTHÈSE

Ce travail, qui est fondamentalement décrit dans la thèse de Ludovic Jacomme, porte sur l'utilisation de méthodes formelles pour l'analyse sémantique de descriptions VHDL synchrones en vue de la synthèse [Jac99]. J'ai participé, avec Alain Greiner, à l'encadrement de cette thèse.

La remarque initiale à partir de laquelle se sont développées les idées de Ludovic Jacomme est que les outils de synthèse logique au niveau transfert de registres disponibles dans l'industrie se basent tous, sans exception, sur une identification syntaxique des éléments mémorisants et des barrières trois états. En ce sens, la sémantique du langage VHDL n'est pas utilisée en tant que telle pour décrire ces éléments. Cela mène à une situation paradoxale dans laquelle des descriptions sémantiquement synthétisables ne sont pas acceptées et, *a contrario*, des descriptions théoriquement non synthétisables sont acceptées. Parfois même un élément est identifié à la place d'un autre, tel un multiplexeur à la place d'une bascule sur état ou *vice-versa*. De plus, les règles d'écriture sont différentes suivant les outils de synthèse, ce qui nuit grandement à la portabilité des descriptions. L'objectif de VHDL, qui est au départ un langage de modélisation, d'être également un standard pour la synthèse n'est pas atteignable en l'état.

Certaines restrictions sont légitimes, telles que celles visant à autoriser uniquement des types et des fonctions ayant une sémantique matérielle bien définie, ou à assurer la possibilité d'une étude locale en garantissant qu'un processus puisse être analysé indépendamment des autres et sur un seul delta-cycle. D'autres n'existent que pour rendre triviale la phase d'identification des registres.

Les algorithmes mis en œuvre dans les outils industriels identifient les registres à l'aide des critères suivants :

- un symbole est de type combinatoire si et seulement s'il est affecté sur tous les chemins d'exécution d'un processus ;
- un symbole représente la sortie d'un élément mémorisant si et seulement s'il n'est pas affecté sur tous les chemins d'exécution. De plus les outils imposent une structure particulière pour reconnaître la condition d'écriture dans le registre ou les conditions de mise à zéro ou à un.

Ces critères partitionnent bien les symboles en deux ensembles, mais ils ne permettent pas une identification correcte des éléments mémorisants.

Ces critères sont naïfs car, *primo*, les chemins sur lesquels le symbole n'est pas affecté peuvent être des chemins qui ne sont jamais empruntés et, *secundo*, un symbole peut être affecté sur tous les chemins, mais avoir quand même un comportement de registre, en fonction du comportement des variables à partir desquelles il est affecté. Les deux contre-exemples ci-après illustrent la violation de ces règles.

```
PROCESS( CK )
BEGIN
  IF (CK='1')
  THEN Q <= D;
  ELSE Q <= Q;
  END IF;
END PROCESS;
```

Q n'est pas combinatoire, car si CK vaut 0, Q conserve sa valeur.

```
PROCESS( A, B, SEL )
BEGIN
  IF (SEL='1')
  THEN S <= A;
  ELSIF (SEL/='1')
  THEN S <= B;
  END IF;
END PROCESS;
```

S n'est pas un registre, car il est affecté sur tous les chemins, même en l'absence de ELSE.

Nous pouvons donc conclure qu'il n'existe pas de règles simples et purement syntaxiques pour identifier correctement les éléments mémorisants.

La méthode d'analyse sémantique conçue et développée par Ludovic Jacomme transforme le code VHDL en un modèle formel basé sur les réseaux de Petri à jeton unique, XVPN, et qui fondamentalement représente un graphe de contrôle et de données (*Control-Data Flow Graph*). Ce modèle est une extension du modèle VPN développé par Emmanuelle Encrenaz [Enc95] et Rajesh Bawa [Baw96] dans le cadre de leurs travaux sur la preuve formelle de matériel.

Des techniques de réductions locales et globales sont utilisées pour mettre en évidence les dépendances de données réelles à l'intérieur d'un processus. Ces techniques consistent à transformer des structures de contrôles en opérations sur les données. Il faut d'une part substituer les variables dans les expressions du code afin de mettre en évidence les dépendances de données, et d'autre part introduire des variables intermédiaires lors de prises de décisions pour obtenir une suite d'affectations uniques, c.-à-d. telles que le membre de gauche soit affecté une unique fois. Pour cela, on introduit une variable pour représenter la valeur du pilote d'un signal. Puis on introduit autant de variables intermédiaires particulières qu'il y a de différentes valeurs prises par une même variable.

Le résultat de ces transformations est un graphe dont la structure est canonique et dans lequel chaque symbole est affecté une unique fois sur tous les chemins d'exécution.

Chaque affectation est ensuite analysée afin de détecter les mémorisations. Pour ce faire, un arsenal symbolique basé sur les diagrammes de décision binaires (BDD) est utilisé pour mettre en évidence les dépendances entre symboles. Il permet pour chaque assignation de déterminer si elle est synthétisable, si c'est un registre, et dans ce cas extrait la donnée, la condition d'écriture dans le registre, les forçages à zéro et à un et s'ils sont asynchrones ou synchrones, ainsi que le front actif de l'horloge. On vérifie ensuite la cohérence de ces assignations avec la liste de sensibilité du processus, et l'on conclu sur la faisabilité de la synthèse et sur l'élément reconnu.

La technique est étendue aux processus possédant plusieurs points de suspension pour extraire des machines à états. Cette extension généralise la méthode, mais apporte peu en pratique car de telles descriptions sont peu utilisées par les concepteurs.

L'utilisation de ces techniques sur un grand nombre de descriptions VHDL a prouvé expérimentalement leurs capacités à analyser correctement les problèmes respectant les hypothèses de départ. Ces mêmes expérimentations ont mis en évidence les lacunes des outils industriels, qui, si l'on sort du cadre strictement syntaxique qu'ils imposent, produisent des résultats aberrants vis-à-vis de la sémantique du langage et ne permettent pas l'utilisation de différentes plates-formes de synthèse sans réécriture du VHDL source. Ludovic Jacomme a développé l'outil `vasy` qui implante la méthode proposée. Cet outil permet d'assurer effectivement la portabilité du VHDL RTL sur les 5 plates-formes industrielles de synthèse auxquelles il a eu accès.

Sur ce sujet, deux articles [JPB98, JPB99] ont été publiés dans deux conférences internationales.

2.5 PLOT D'ENTRÉE/SORTIE TROIS ÉTATS POUR BUS PCI

Sur un aspect très différents, la conception de circuit, j'ai participé avec Franck Wajsbürt et Karim Dioury à l'étude et à la réalisation d'un plot d'entrée/sortie pour bus PCI – plot utilisé notamment dans le circuit PCIDDC développé par le LIP6 –.

L'objectif du travail a été de concevoir le schéma et les masques d'un plot d'entrée/sortie unique compatible avec les deux standards PCI pour des alimentations sous 3.3 V et 5.0 V. Ces standards imposent fondamentalement des contraintes sur la puissance et la pente des signaux. En conséquence, nous avons été amené à définir un schéma d'amplificateur permettant de faire varier ces contraintes indépendamment l'une de l'autre.

L'avantage principal de cette indépendance est qu'il est possible de faire des implantations avec des choix différents de consommation et de performance en faisant uniquement varier la taille des transistors, tout en respectant la norme.

Une contrainte supplémentaire que nous nous sommes imposés est l'utilisation de l'approche « symbolique » d'**Alliance**, décrite en annexe de ma thèse [Pét94], pour le dessin des masques, approche qui permet automatiquement de cibler différents processus de fabrication.

Ce travail a donnée lieu à publication dans une conférence internationale [WDP97] avec comité de programme et dans une revue internationale [WPD98] avec comité de lecture.

2.6 APPORTS D'**ALLIANCE**

Les travaux effectués autour de la chaîne de CAO **Alliance** ont eu un impact considérable pour notre équipe. Ils ont tout d'abord fédérés les chercheurs en CAO et en conception autour d'un projet commun et ont permis l'expérimentation à large échelle de la plateforme. Les résultats tangibles sont les suivants :

- il y a aujourd'hui une bonne trentaine d'outils de CAO stables développés autour des structures de données d'**Alliance**. Ces outils sont quotidiennement utilisés par les chercheurs et étudiants de l'équipe ASIM du LIP6, mais aussi dans d'autres laboratoires, notamment à des fins d'enseignement ;
- des circuits conçus en grande partie avec **Alliance** ont été fabriqués ou sont en cours de fabrication. Certains de ces circuits sont peu complexe, tel le générateur de mémoire morte d'**Alliance**[DGP95], qui est utilisé par la société Atmel aux USA. D'autres le sont beaucoup plus. Les derniers en date, Rcube et PCIHSL, ont été effectivement utilisés dans la machine parallèle MPC développée au LIP6, avec comme resultat un transfert de ces technologies vers la société Tachys.

  Un autre exemple est le circuit SPIN, un réseau sur puce, qui utilise l'approche des masques portables et la génération procédurale pour le placement et le routage *custom* pour viser la technologie la plus récente disponible auprès de la compagnie STMicroelectronics ;
- les structures de données de base d'**Alliance** sont utilisées industriellement par la société Avertec (une spin-off du département).

Lors de la diffusion d'**Alliance** s'est posée la question de la bonne manière de procéder. J'ai milité depuis le début, et avec d'autres, pour le principe d'une distribution gratuite sous licence GNU. Le présent semble nous avoir donné raison puisque une telle distribution n'a pas nuit aux possibilités de transfert technologique et a apporté une certaine notoriété à notre équipe, du moins en ce qui concerne notre capacité à mener à bien des projets conséquents. *A contrario*, certains chercheurs, désireux d'exploiter industriellement leurs résultats, n'ont pas voulu distribuer leurs outils au sein d'**Alliance** et ont nécogiés licences auprès de l'université pour la

valorisation industrielle. Deux cas se presentent alors.

1. la licence est exclusive. Dans ce cas, le résultat pratique est que notre équipe ne travaille plus sur le problème (cas de l'outil TAS d'Avertec) ;
2. la licence n'est pas exclusive. Les chercheurs du laboratoire peuvent continuer à travailler sur ces outils, dans le but de les améliorer. Ils devront négocier une licence avec l'université pour les valoriser, ce qui en pratique exclue une diffusion « libre ». En pratique, il y a donc redéveloppement d'une infrastructure pour continuer à travailler sur un sujet (cas de l'outil YAGLE d'Avertec).

L'expérience acquise a d'ailleurs été bénéfique au sein de notre équipe, puisque les travaux dans le domaine de la CAO actuellement initiés envisagent le même type de valorisation. Par exemple les travaux en placement-routage actuellement en cours se fondent sur une plateforme développée par R. Escassut et C. Masson (Bull), qui ont fait le choix d'une distribution sous licence GNU.

Dans le même ordre d'idée, un outil de génération de structures développé au LIP6 et dont les auteurs, C. Douady et B. Boutillier, partis dans l'industie, ont néanmoins souhaité distribuer sous licence GNU cet outil, également utilisé dans le département.

Nous assistons en fait, grâce à ces distributions libres, à une sorte d'externalisation des compétences qui permet à l'équipe d'avoir plus de forces pour cibler les problèmes de recherche particuliers qui l'intéresse, l'accès aux sources étant un gage de pérénité.

2.7 PROTOTYPES LOGICIELS

Ludovic Jacomme a développé l'outil de synthèse par analyse sémantique du VHDL **vasy** qui implante les techniques de reconnaissance sémantique., J'ai contribué à ce développement en écrivant le traducteur vers Verilog. Cet outil est utilisé en *front-end* des outils de simulation et de synthèse VHDL d'`Alliance` qui ont une syntaxe d'entrée comprenant uniquement le sous-ensemble concurrent (description de réseaux booléens) de VHDL. C'est en ce sens un pas important pour élargir l'accès aux outils d'**Alliance** aux concepteurs habitués à un VHDL plus étendu.

On peut noter que le générateur de ROM **grog** [DGP95], développé en partie durant ma thèse, est utilisé par la société ATMEL en production.

Ces outils sont disponibles dans la chaîne de CAO **Alliance**.

L'activité autour d'**Alliance** est fédératrice au sein de notre équipe, et nous en voulons pour preuve que la version 5.0, qui est nouvellement disponible, est principalement lié à la volonté de nouveaux doctorants contributeurs d'outils, principalement Christophe Alexandre et François Donnet. Notre contribution dans ce contexte est de pousser à la mise en œuvre dans cette nouvelle distribution d'outils plus modernes pour la compilation multi-plateforme et la génération des scripts de compilation. Ceci devrait faciliter la maintenance, les distributions futures et le développement de nouveaux outils, et assurer, nous l'espérons, la pérennité d'**Alliance**.

2.8 BIBLIOGRAPHIE

[Enc95] Emmanuelle Encrenaz. *Une méthode de vérification de propriétés de programmes VHDL basée sur des modèles formels de réseaux de Pétri.* Thèse de Doctorat, Université Paris 6, décembre 1995.

[Baw96] Rajesh K. Bawa. *Un environnement intégré pour la vérification formelle et l'analyse de programmes VHDL.* Thèse de Doctorat, Université Paris 6, décembre 1996.

[Pét94] Frédéric Pétrot. *Outils d'aide au développement de bibliothèques VLSI portables*. Thèse de Doctorat, Université Paris 6, juillet 1994. Annexe 1, coécrite avec Franck Wajsbürt.

2.9 PUBLICATIONS RELATIVES AU CHAPITRE 2

N'apparaissent ici que les publications postérieures à la soutenance de ma thèse.

Journaux scientifiques internationaux avec comité de programme

[WPD98] Franck Wajsbürt, Frédéric Pétrot, and Karim Dioury. Transistor controlled slew rate process independent PCI compliant I/O buffer with possible power/delay trade-off. *Microelectronics journal*, 29 :733–740, October 1998.

Conférences internationales avec comité de programme

[GPP94] Alain Greiner, François Pêcheux, and Frédéric Pétrot. Symbolic debugger for the design of portable module generators. In *IX Congresso da SB Micro*, pp. 690–698, Gramado, Brazil, August 1994.

[GJPW94] Alain Greiner, Ludovic Jacomme, Frédéric Pétrot, and Franck Wajsbürt. Yet another rectangle data structure for VLSI CAD algorithms. In *6th International Conference on Microelectronics*, pp. 104–107, September 1994.

[GP94a] Alain Greiner and Frédéric Pétrot. A public domain high performances portable ROM generator. In *20th Euromicro*, pp. 414–419, Liverpool, England, IEEE, September 1994.

[GP94b] Alain Greiner and Frédéric Pétrot. Using C to write portable CMOS VLSI module generators. In *The European Design Automation Conference*, pp. 676–681, Grenoble, France, September 1994.

[GPW95] Alain Greiner, Frédéric Pétrot, and Franck Wajsbürt. Fixed grid symbolic layout translation into mask rectangles. In *2nd International Workshop on Mixed Design of Integrated Circuits and Systems*, pp. 281–286, Krakòw, Poland, May 1995.

[DGP95] Marcelo Duhalde, Alain Greiner, and Frédéric Pétrot. A high perfomance modular embedded ROM architecture. In *1995 IEEE International Symposium on Circuits and Systems*, pp. 1057–1060, Seattle, USA, April 1995. IEEE.

[GPW96] Alain Greiner, Frédéric Pétrot, and Franck Wajsbürt. A portable layout approach well adapted to education. In Georges Kamarinos, Nadine Guillemot, and Bernard Courtois, editors, *Proceeding of the European Workshop « Microelectronics Education »*, pp. 97–102, Grenoble, France, 1996. World Scientific Publishing Co.

[WDP97] Franck Wajsbürt, Karim Dioury, and Frédéric Pétrot. Low power, process independent, full transistor controlled slew rate, PCI compliant I/O pads. In *Proceeding of the 21st International Conference on Microelectronics*, Volume 2, pp. 811–814, Niš, Yougosliavia, September 1997. IEEE.

[DPJ97a] Julien Dunoyer, Frédéric Pétrot, and Ludovic Jacomme. Intrinsic limitations of logarithmic encodings for low power finite state machines. In *4th International Workshop on Mixed Design of Integrated Circuits and Systems*, pp. 613–618, Poznan, Poland, June 1997.

[DPJ97b] Julien Dunoyer, Frédéric Pétrot, and Ludovic Jacomme. Limitations of logarithmic encodings for low power finite state machines. In *International Conference on Electronics, Circuits and Systems*, pp. 522–528, Cairo, Egypt, December 1997.

[JPB98] Ludovic Jacomme, Frédéric Pétrot, and Rajesh K. Bawa. Formal extraction of memorizing elements for sequential vhdl synthesis. In *24th Euromicro*, pp. 317–320, Vesteras, Sweden, August 1998. IEEE.

[JPB99] Ludovic Jacomme, Frédéric Pétrot and Rajesh K. Bawa. Formal analysis of single wait vhdl processes for semantic based synthesis. In *12th International Conference on VLSI Design*, pp. 151–156, Goa, India, January 1999. IEEE.

Conférences nationales avec comité de programme

[JP96] Ludovic Jacomme and Frédéric Pétrot. Du codage des automates en vue de la synthèse sur silicium : comparaison systématiques et critique des approches existantes. In *2ème Conférence Nationale sur la résolution pratique de problèmes NP-complets*, pp. 293–302, Dijon, France, mars 1996. Tacoma Ed.

[DPJ97c] Julien Dunoyer, Frédéric Pétrot, and Ludovic Jacomme. Stratégie de codage des automates pour les applications basse-consommation : expérimentation et interprétation. In *Journées d'études faible tension faible consommation*, pp. 145–152, Paris, France, novembre 1997.

2.10 PARTICIPATION À L'ENCADREMENT DE THÈSES

[Jac99] Ludovic Jacomme. *Analyse sémantique de descriptions VHDL synchrones en vue de la synthèse*. Thèse de l'UPMC soutenue en octobre 1999. En codirection avec Alain Greiner.

Deuxième partie

Systèmes intégrés

Chapitre 3

Introduction

Depuis 1996, les travaux que j'ai menés ont porté sur les problèmes liés à la conception de systèmes intégrés sur une unique « puce », les *Systems on Chip*. Cette dénomination, bien que couramment admise, est ambiguë. En effet, par le terme « système » on entend généralement en informatique un matériel prédéfini sur lequel s'exécutent des applications très variées. Dans le cas qui nous intéresse, on part d'une application unique et bien spécifiée et on cherche à la réaliser à la fois en logiciel et en matériel sous contrainte de coût, puissance consommée, performance, etc., sans idées préconçues sur la nature logicielle ou matérielle de telle ou telle partie. C'est ce qui fait l'originalité de ces méthodes par rapport aux méthodes plus classiques en informatique. Pour ce type de problèmes, nous préférons donc le terme d'application intégrée.

La définition de méthodes n'a de sens que si l'on peut les confronter à l'expérimentation. Je me suis donc également intéressé à la réalisation pratique de ses systèmes en spécifiant et développant, avec l'équipe dont j'ai la charge, des outils pour un certain nombres d'étapes identifiées de la conception. La totalité des réalisations constitue un ensemble cohérent pour la conception et la réalisation de systèmes sur puce. Nous avons nommé cet ensemble **Disydent**[APH01].

3.1 Problématique générale

La conception d'applications intégrées pose un certain nombre de problèmes nouveaux par rapport à la conception de systèmes plus classiques. En effet, la spécification initiale est une application et non un système pensé pour supporter un système d'exploitation et une multitude d'applications utilisant un matériel standard.

Le problème qui se pose alors est d'être capable, à partir d'une spécification initiale, de dérouler un flot de conception nécessitant aussi peu de réécriture des tâches que possible. Ce flot n'a pas l'ambition d'être totalement automatique : il nécessite au contraire l'intervention du concepteur pour faire les choix stratégiques.

La spécification comporte une partie fonctionnelle, qui décrit le comportement, et une partie non-fonctionnelle qui fournit les contraintes externes à respecter. La spécification fonctionnelle initiale, c.-à-d. séquentielle, ne peut généralement être utilisée telle quelle pour une implantation mixte en matériel et logiciel. Il faut donc disposer d'un langage de spécification supportant le parallèlisme à gros grain avec une sémantique d'exécution et de communication non-ambiguë et s'assurer que ce langage et sa sémantique pourront être respectés et implantés par la suite. Ceci ne veut pas dire qu'il ne sera pas nécessaire de réécrire une tâche pour en faire l'implantation matérielle par exemple. Par contre, cette tâche une fois réécrite, par exemple pour

optimiser ses performances ou pour la rendre synthétisable, pourra être substituée telle quelle dans la spécification parallèle, car sa sémantique sera identique même si son implantation à changé. Les problèmes à résoudre sont abordés un par un dans la suite.

– La première phase de la conception consiste à modéliser l'application que l'on cherche à intégrer au niveau fonctionnel. Nous nous sommes arrêté, assez classiquement, à un modèle de comportement faisant apparaître le parallélisme à gros grain entre tâches [Kah74, Hoa85], comme illustré sur la figure 3.1. Le découpage d'une spécification séquentielle en

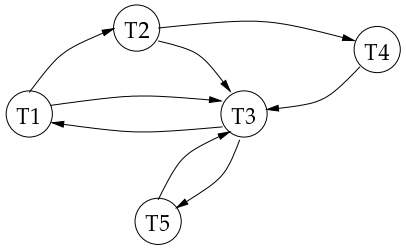

FIG. 3.1 – Une application décrite comme un graphe de tâches communicantes.

tâches est un travail de concepteur de systèmes, lequel doit définir la bonne granularité de chaque tâche et les données à échanger entre tâches. Ce modèle permet, en effet, de mettre en évidence le gain que l'on aura en réalisant une tâche d'une manière ou d'une autre, car on peut avoir d'une part l'accélération fournie par telle ou telle implantation et d'autre part le recouvrement d'exécution des différentes tâches entre elles et avec les flux de données. L'utilité de ce modèle découle directement de la loi d'Amdahl [Amd67], qui nous indique que seule l'utilisation du parallélisme effectif peut permettre un gain significatif en performance pour une application.

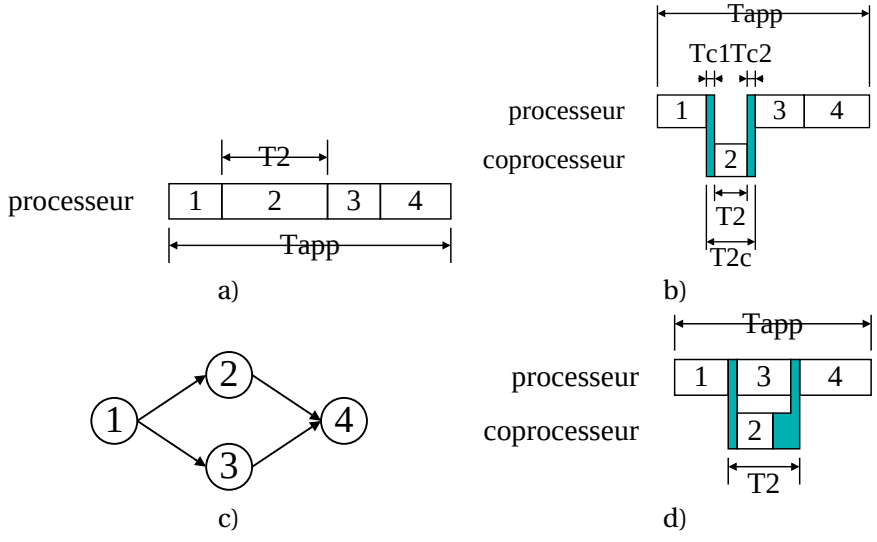

FIG. 3.2 – Illustration de la loi d'Amdahl

La figure 3.2 illustre en pratique les limitations qu'impose la loi d'Amdahl. En a), nous avons la spécification séquentielle. En b), une partie de la spécification est exécutée sur un accélérateur matériel, donc plus rapidement, mais l'on paye un coût pour la communication qui rend le gain négligeable ou même ralentit le système. En effet, Amdahl nous enseigne que le temps maximum que l'on peut gagner est le temps de la partie que l'on a accélérée. Si on fait apparaître explicitement les dépendances de données, comme en

c), alors on a recouvrement des temps d'exécution des différentes tâches mais aussi des temps de communications, si le matériel le permet, comme cela est illustré en d).
La simulation du graphe de tâches permet de valider fonctionnellement l'application et d'évaluer la complexité de chaque tâche pour en déduire un temps de calcul suivant qu'elle est réalisée en matériel ou en logiciel. La simulation du graphe permet aussi de mesurer les communications entre les tâches en terme de volumes de données transférées sur chaque canal. Ceci permet de déterminer le coût en communication impliqué par un changement de nature d'une tâche de logiciel en matériel ou inversement en fonction de la nature des tâches avec lesquelles elle communique. La spécification parallèle doit donc être exécutable.
J'ai mené ces travaux liés à la spécification fonctionnelle en collaboration avec Denis Hommais au LIP6, et en m'appuyant sur les travaux de nombreux collègues du laboratoire de recherche de Philips à Eindhoven [dKES+00].
La sémantique des communications entre les tâches ainsi qu'un modèle d'exécution de la spécification sont présentés dans le chapitre 4 ;

– La seconde étape consiste à implanter la spécification. Il faut considérer cinq sous-parties :

  1. la définition de l'architecture générale supportant l'implantation. Cette base architecturale peut être initialement relativement générique. Elle doit néanmoins définir clairement les types de processeurs supportés, les services fournis par le ou les noyaux de système d'exploitation embarqués, les types d'interconnexions supportés, la manière d'assurer les accès atomiques aux ressources, etc. Cette base architecturale est dénommée « plate-forme », et restreint les choix d'implantation. C'est aussi une nécessité si l'on veut être capable d'avoir un chemin relativement automatique vers le matériel.

     Il s'agit ici de déterminer les caractéristiques générales de l'architecture cible :
     – nombre et type de processeurs ;
     – type du ou des noyaux logiciels ;
     – nombre et type des coprocesseurs ;
     – organisation des interconnexions.

     Ce dernier point est particulièrement important car il définit les capacités du système en bande passante et donc son adéquation à une classe d'applications. La figure 3.3 illustre deux implantations totalement différentes d'une même architecture générique.

     Le choix de l'architecture et le partitionnement ne peuvent pas être fait séparément selon [KM98]. Le choix du bus de communication, par exemple, dépend du débit qui est nécessaire aux différents éléments pour communiquer et des tâches affectées à ces composants. À l'inverse, suivant le débit du bus choisi, l'affectation de tâches aux éléments du système sera ou non réalisable. L'idée derrière la définition d'une plate-forme est précisément de briser – ou du moins d'alléger – cette interdépendance en faisant des choix généraux d'architecture tôt dans la conception.

     « L'approche de la conception à base de plate-forme est en quelque sorte l'antithèse de la co-conception matérielle/logicielle ! » selon Wayne Wolf [Wol01]. En effet, ce choix limite les degrés de libertés de la conception. Les plates-formes dans lesquelles l'essentiel des acteurs sont des processeurs programmables se développent car elles permettent la souplesse du logiciel. Concevoir des multiprocesseurs *on-chip* rend la conception de **SoCs** proche de celle des systèmes classiques.

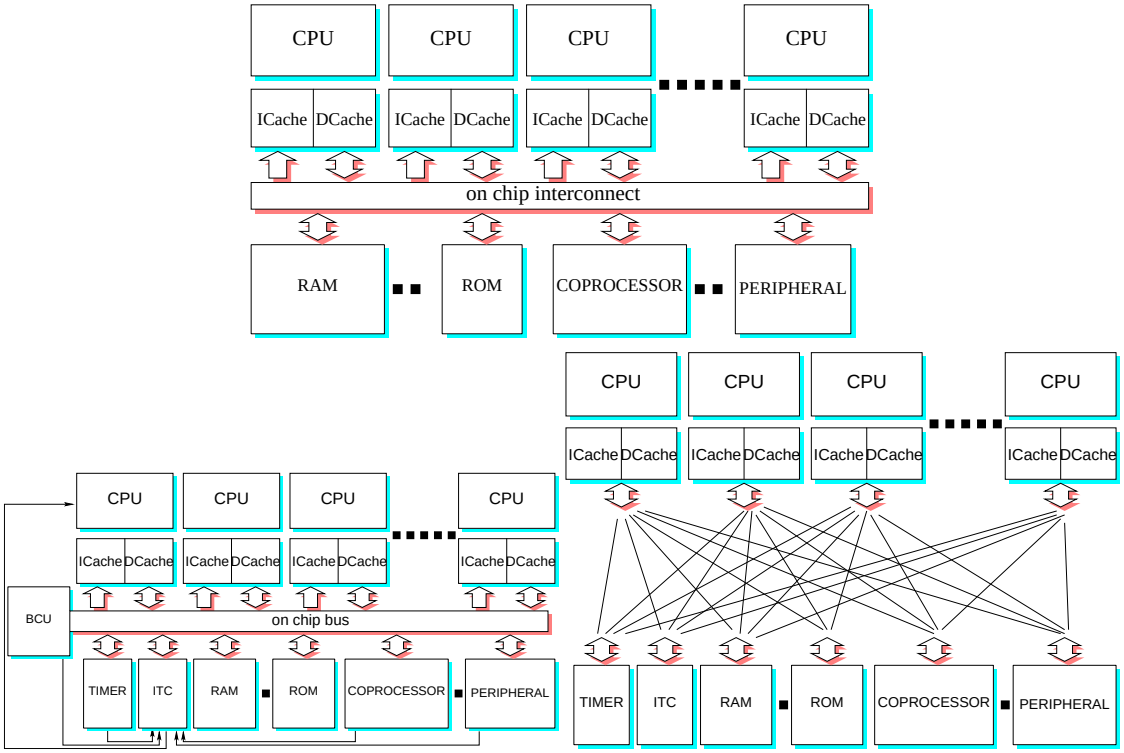

FIG. 3.3 – Choix de l'architecture de communication du système.

Cependant, une plate-forme est une architecture « molle », et ne serait-ce que pour des contraintes liées à la puissance consommée, le câblage d'algorithmes qui seront réalisés par un coprocesseur matériel spécialisé est encore nécessaire ;

2. le choix d'exécuter une tâche soit sur un coprocesseur matériel, soit en logiciel sur un processeur généraliste.

   On décide pour chaque tâche si elle sera réalisée en matériel ou en logiciel et par quel processeur ou coprocesseur du système elle sera réalisée, comme illustré sur la figure 3.4. Les choix de cette étape sont faits par le concepteur en fonction des spécifications non-fonctionnelles de l'application, des composants disponibles et d'une évaluation précise du graphe des tâches pour obtenir la complexité de chaque tâche et le volume de données échangées entre les tâches. Plusieurs tâches peuvent être regroupées dans un même élément de traitement. De même, dans le cas d'un coprocesseur la réutilisation du même matériel pour effectuer plusieurs tâches est une question de compromis entre parallélisme et surface du système résultant.

   Il n'est, selon moi, pas souhaitable d'automatiser ce choix. En effet, s'il existe des critères objectifs tels la performance ou la puissance consommée, il est beaucoup plus difficile d'apprécier le gain en flexibilité apporté par le logiciel – mise au point tardive, correction *a posteriori* de bugs, voire modification de la fonctionnalité –. La validité des choix effectués à cette étape ne peut être vérifiée qu'après l'étape suivante qui réalise les communications.

   Pour décider d'une implantation, le concepteur doit disposer d'outils lui permettant de mesurer la pertinence de ses décisions. Le niveau d'abstraction de tels outils est une question en soit. J'ai très tôt fait le choix, afin de garantir l'exactitude des résultats de simulation, d'orienter mon travail vers une modélisation au cycle près des com-

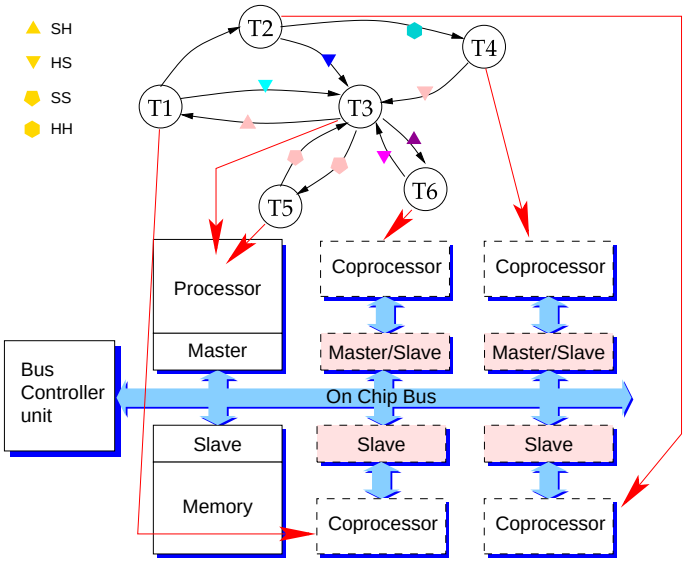

FIG. 3.4 – Affectation des tâches.

posants cibles. Ainsi peut-on, avec un simulateur précis au cycle, avoir une image exacte des performances d'une certaine architecture. Cette modélisation exige un simulateur cycle par cycle, ainsi que des modèles de comportement avec cette précision. J'ai donc défini une technique de simulation et développé, avec l'aide précieuse de Denis Hommais, un simulateur et un certain nombre de composants. Les enjeux et résultats de ce travail sont présentés aux chapitre 5 ;

3. la synthèse des communications, qui consiste à faire communiquer effectivement deux tâches entre elles, qu'elles soient matérielle ou logicielle. Cette synthèse repose sur les choix des sémantiques de communication qui ont été fait au niveau fonctionnel, et fait intervenir du logiciel et du matériel spécialisé.

   Dans le graphe représentant l'application, les tâches échangent des messages par des canaux de communication représentés par les arcs. Une fois les tâches affectées aux ressources du système, il faut adapter ces échanges de données aux supports physiques effectivement disponibles pour la communication (bus, lignes d'interruptions, communications point-à-point comme une fifo matérielle dédiée entre deux coprocesseurs, mémoires, ...) et à la nature matérielle ou logicielle de chaque tâche.

   Il faut remplacer les primitives utilisées par les tâches pour communiquer par les méthodes matérielles ou logicielles du système qui sont le support effectif des communications. Les méthodes d'échange de données et de synchronisation qui peuvent être mises en œuvre sont nombreuses : mémoire partagée ou files de messages pour les données, interruptions ou scrutations pour la synchronisation.

   Ce travail, effectué la encore avec Denis Hommais a bénéficié d'une collaboration étroite avec Jean-Yves Brunel et Wido Kruytzer du laboratoire de recherche de Philips à Eindhoven, dans le cadre du projet Esprit COSY.

   L'approche automatique que Denis Hommais et moi avons définie et mise en œuvre est présentée dans le chapitre 6 ;

4. la synthèse de coprocesseurs spécialisés. Cette approche a été essentiellement définie par Ivan Augé, qui, depuis sa thèse il y a une douzaine d'années, défend l'idée que les outils de synthèse de haut niveau ne doivent pas effectuer d'allocation ni d'assi-

gnation de ressources, mais essentiellement générer un contrôleur contraint par la fréquence d'utilisation. Le travail d'un concepteur de coprocesseur consiste à définir le degré de parallélisme de son architecture, ce qui fondamentalement revient à définir grossièrement le chemin de données. Cette expertise architecturale est, avec le comportement haut niveau et la fréquence d'exécution du circuits, l'entrée du processus de synthèse. L'approche retenue est détaillée au chapitre 7 ;

5. la synthèse du logiciel, qui consiste à compiler la tâche et assurer le bon déroulement des communications grâce à l'utilisation de pilotes spécifiques. La compilation du logiciel embarqué pose un certain nombre de problèmes, en particulier dans le cas des multiprocesseurs, comme, par exemple, la prise en compte de la migration des tâches, le placement des données, etc. Mais ces problèmes sortent du cadre de notre travail. Par contre, comme la spécification est parallèle, il faut sur le ou les processeurs embarqués un noyau permettant d'exécuter un ensemble de tâches. J'ai été amené à spécifier, là encore avec Denis Hommais, et à développer un micro-noyau multiprocesseurs et multi-tâches, et nous le présentons au chapitre 8. Je présente également brièvement une implantation de la bibliothèque de communication sur ce micro-noyau réalisée par Denis Hommais.

Définition des niveaux de transaction

Les transformations effectuées pour franchir les différents niveaux d'abstraction lors de l'implantation se font à sémantique constante, aussi bien pour ce qui concerne la fonctionnalité que pour les communications. La réutilisation d'algorithmes ou de circuits peut se faire à tous ces niveaux si la sémantique de communication est respectée.

On définit ainsi des niveaux de transactions, qui sont les suivants :

Applicatif : à ce niveau, seules les informations fonctionnelles sont significatives. Les algorithmes particuliers utilisés, etc, ne sont pas encore définis. Les temps d'exécution et l'ordonnancement des tâches ne sont pas visibles par exemple ;

Système : La taille des tampons dans les communications est définie. La nature matérielle ou logicielle d'une tâche n'est pas encore décidée ;

Plate-forme : l'implantation effective des tâches est décidée. Le protocole de communication bas niveau est défini, mais pas son implantation effective, qui peut être un bus ou un réseau commuté par exemple. De plus on peut très bien imaginer à ce niveau là l'utilisation de *wrappers* permettant de traduire un protocole dans le protocole de la plate-forme. Les adresses permettant d'identifier les différents acteurs sont nécessaires. Les données sont découpées en mots et paquets acceptables par le support physique de la communication ;

Physique : la nature de l'interconnexion est fixée, les types d'arbitrages, la manière d'implanter les opérations atomiques, d'assurer la cohérence des caches, etc, est définie.

Ce découpage est principalement la conclusion d'une expérience industrielle menée par Philips. Ce travail a donné lieu à deux publications auxquels nous sommes associés dans des conférences internationales [BWLP99, BKK+00].

3.2 MÉTHODE

Le scénario de conception, établi avec Ivan Augé et Denis Hommais, est présenté graphiquement sur la figure 3.5.

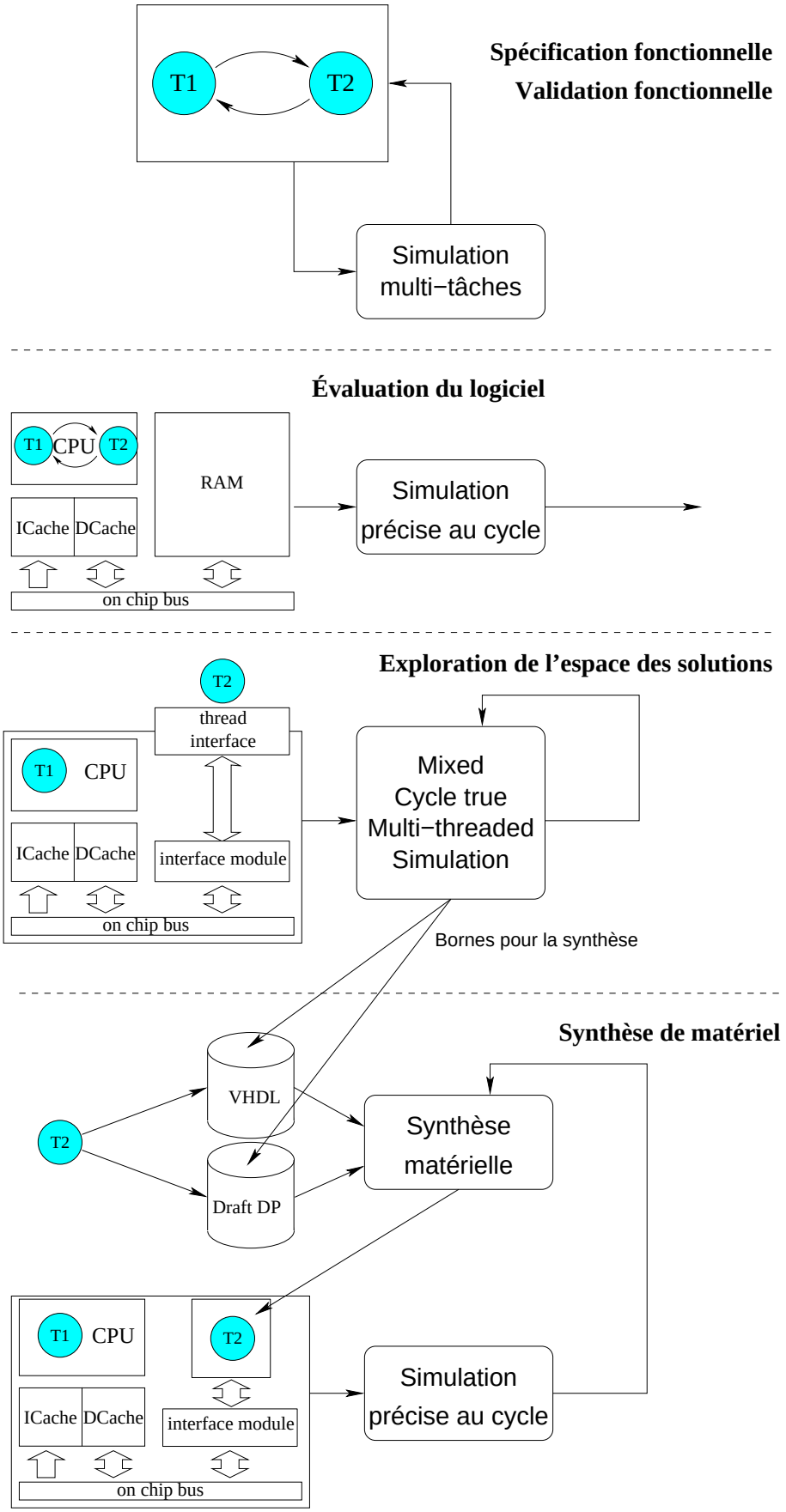

FIG. 3.5 – Scénario de conception de systèmes intégrés

L'idée repose sur l'utilisation de primitives de communications pour lesquelles on dispose d'implantations logicielles et matérielles « portables ».

En ce qui concerne le logiciel, l'équipe à fait le choix d'utiliser sur le standard POSIX [] pour la création et l'exécution de tâches. Ce standard défini un petit ensemble de fonctions disponibles sur quasiment tous les systèmes d'exploitation.

Ce choix d'implantation très tôt dans la méthode de conception est dicté par le coté pragmatique de l'approche, qui cherche à minimiser le code à écrire et réécrire pour l'adaptation à de nouveau systèmes.

Ainsi, si le concepteur décrit chacune des tâches de l'application sous forme d'une tâche POSIX, il peut compiler et exécuter l'application directement sur la machine hôte de son choix.

Si par ailleurs le processeur embarqué exécute un noyau POSIX, ce qui est le cas de la plupart des offres commerciales – VxWorks, QNX, LynuxWorks (ex LynxOS), Chorus, eCos – , alors il suffit de faire une compilation croisée de l'application et de l'exécuter sous un simulateur comprenant le processeur, ses caches, un bus et de la mémoire. Le problème de l'interaction avec le monde extérieur est résolu en utilisant des fichiers sur la machine exécutant le simulateur. Cette manière de procéder doit néanmoins permet de spécifier des contraintes non-fonctionnelles.

La première étape consiste à exécuter la spécification fonctionnelle parallèle sur une machine hôte.

Lorsqu'elle est valide, l'application est compilée grâce à un compilateur croisé pour le processeur embarqué. À condition de disposer d'un simulateur précis au cycle, cette seconde étape permet de connaître très précisément les performances de l'application s'exécutant de manière purement logicielle, en fonction des ressources disponibles sur le ou les processeurs embarqués.

À l'aide des informations recueillies lors de la simulation cycle, le concepteur peut décider d'utiliser un ou plusieurs coprocesseurs dédiés à l'application. Afin d'expérimenter facilement, il doit d'abord être capable d'utiliser une tâche telle que décrite pour la spécification et la faire apparaître comme un élément matériel du point de vue de la simulation. Un tel « adaptateur » possède un comportement précis au cycle pour le simulateur cycle et un comportement fonctionnel vis-à-vis de la tâche. Pour cela, il dispose d'un modèle de simulation, le *threader*, paramétrable en débit et en latence, qui assure la communication entre le simulateur cycle et une ou plusieurs tâches POSIX s'exécutant sur l'hôte.

Ensuite, il doit disposer d'un moyen simple pour décrire la manière dont se comportent les canaux de communications – cette phase est l'objet du chapitre 6 du présent document –. Il peut alors mesurer les performances de son système en fonction des performances attendues des tâches.

L'exploration de l'espace de conception consiste alors à modifier l'affectation des tâches (sur matériel ou logiciel) et à modifier la manière d'effectuer les communications. La simulation permet d'orienter le concepteur vers des choix sensés, évitant ainsi un trop grand nombre de simulations.

Arrive finalement l'implantation dans laquelle l'architecture matérielle de chaque tâche est suffisamment raffinée pour pouvoir en déduire un modèle de simulation au cycle près. Des lors, l'exécution de l'application se fait entièrement sous le simulateur cycle et fournit la performance du système.

Cette méthode à été publiée dans une conférence nationale [HPA00] et une conférence internationale [APH01].

3.3 BIBLIOGRAPHIE

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, pp. 471–475, Stockolm, Sweden, August 1974.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[KM98] Peter Voigt Knudsen and Jan Madsen. Communication estimation for hardware/software codesign. In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign (Codes/CASHE)*, 1998.

[dKES+00] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers. Yapi : Application modeling for signal processing systems. In *Proceedings of the 37th Design Automation Conference*, pp. 402–405. IEEE, June 2000.

[Wol01] Wayne Wolf. Codesign, looking beyond and forward. In *CODES 01*, Copenhagen, Dennemark, April 2001.

3.4 PUBLICATIONS RELATIVES AU CHAPITRE 3

Conférences internationales avec comité de programme

[BWLP99] Jean-Yves Brunel, Yosinori Watanabe, Luciano Lavagno, and Frédéric Pétrot. Cosy : levels of interfaces for modules used to create a video system on chip. In Jean-Yves Roger, Brian Stanford-Smith, and Paul Kidd, editors, *EMMSEC'99, Buissiness and work in the information society : new technologies and application*, pp. 772–778, Stockholm, Sweden, June 1999. IOS Press.

[BKK+00] J-Y. Brunel, W. M. Kruijtzer, H. J. H. N. Kenter, F. Pétrot, L. Pasquier, E. A. de Kock, and W. J. M. Smits. Cosy communication ip's. In *37th Design Automation Conference*, pp. 406–409, Los Angeles, CA, June 2000.

[APH01] Ivan Augé, Frédéric Pétrot, and Denis Hommais. A pragmatic approach to the design of embedded systems. In *Design Automation and Test in Europe*, pp. 170–174, Munich, Germany, March 2001. IEEE.

Conférences nationales

[HPA00] Denis Hommais, Frédéric Pétrot et Ivan Augé. Une approche pour la conception des systèmes intégrés, In *IIIèmes Journées Nationales du Réseau Doctoral de Microélectronique*, Montpellier, France, Mai 2000, pp. 76-77.

Chapitre 4

Spécification fonctionnelle parallèle exécutable

4.1 Introduction

Nous présentons dans ce chapitre les travaux que nous avons mené autour de la spécification fonctionnelle d'applications en vue de leur intégration sur silicium. À ce niveau, aucune décision sur la façon dont telle ou telle partie de l'application sera réalisée n'est prise, seule la modélisation de la totalité de l'application est concernée.

4.2 Problématique

Lorsqu'il chercher à effectuer l'implantation d'une application en tant que système matériel/logiciel, le concepteur dispose généralement d'une spécification fonctionnelle décrite comme un processus séquentiel unique et de spécifications non-fonctionnelles définissant la latence, le débit, la surface, la puissance consommée, ou tout autre caractéristique liée à l'environnement d'utilisation du système.

Son premier travail consiste à partitionner ce processus unique en un ensemble de tâches séquentielles s'exécutant concurremment et communiquant grâce à des canaux de type FIFO ou de données partagées [Hoa85]. Ceci permet de mettre en évidence le parallélisme à gros grain inhérent à l'application, la synchronisation s'effectuant par des mécanismes variés. Ce travail est ardu, car il faut :

- extraire de la spécification séquentielle les « bonnes variables » de communications [Cal93] permettant de satisfaire les spécifications non fonctionnelles sans engorger les communications ;
- décider de la granularité de chacune des tâches, indépendamment de son implantation future, puisqu'à ce stade le choix entre matériel et logiciel n'est pas fait ;

L'objectif est d'obtenir, à partir de cette spécification, une implantation de manière automatique. Cela nécessite que les communications soient clairement identifiées dans la spécification et qu'elles aient une sémantique bien définie et indépendante de leur implantation.

4.3 Primitives de communication

Comme nos collègues de chez Philips [dKES+00], nous avons retenu le formalisme des « réseaux de Kahn » pour modéliser nos systèmes[Kah74, KM77]. Ce formalisme est bien adapté à la description fonctionnelle des applications multimédias gérant des flux de données.

Dans ce formalisme, un système est décrit comme un ensemble de tâches séquentielles qui communiquent par l'intermédiaire de canaux de type FIFO.

Dans les réseaux de Kahn, un canal FIFO est tel que :

1. il possède un unique producteur et un unique consommateur, et est donc unidirectionnel ;
2. les données émise dans un certain ordre par le producteur sont reçues dans le même ordre par le consommateur, dans un temps borné mais non définit ;
3. il peut contenir une infinité d'éléments.

L'accès à une FIFO s'effectue uniquement grâce à une primitive de lecture et une primitive d'écriture. La lecture et l'écriture sont toujours réalisées lors du retour des primitives. Pour l'écriture, cela ne pose pas de problème vu la taille infinie des FIFOs. Pour la lecture, cela signifie que si une tâche est en attente de données, alors elle cesse de s'exécuter et ne reprendra son exécution que lorsque le nombre de données attendues aura été extrait du canal dans lequel elle consomme. On qualifie une telle lecture de bloquante, car elle peut bloquer – suspendre – la tâche qui l'exécute. On remarque qu'il est impossible de tester la présence d'une donnée dans une FIFO avant de faire une lecture.

Kahn a prouvé que, dans de tels réseaux, l'ordre et la valeur des données circulant dans chacune des FIFOs du système est indépendante de la durée d'exécution de chaque tâche. Cela signifie que puisque la synchronisation entre les tâches est faite par la circulation des données, il n'est pas nécessaire de savoir à chaque instant quelle tâche fait quoi, et l'on n'a donc pas d'ordonnanceur explicite au niveau système. Un premier corollaire intéressant pour nous est qu'indépendamment de la nature logicielle ou matérielle d'une tâche après implantation, on pourra garantir le bon fonctionnement du système si l'on respecte la sémantique de communication. Un deuxième corollaire est que la vitesse relative d'exécution des tâches n'influe que sur la performance.

En pratique, la profondeur des FIFOs doit être bornée. Pour assurer la même sémantique de communication, il faut alors que l'écriture ne prenne place que s'il y a des cases libres dans la FIFO. La primitive d'écriture devient alors bloquante elle aussi. Parks [Par95] a étudié ce modèle dans le cadre de Ptolemy[BHL+94]. Il a prouvé que cette modification conserve les propriétés des réseaux de Kahn, en y ajoutant la possibilité d'apparition d'inter-blocages. Il existe deux types d'inter-blocages :

1. ceux qui sont liés à une erreur dans la spécification et qui amèneraient à remplir infiniment une FIFO dans le modèle initial de Kahn ;
2. ceux qui sont liés à un choix malheureux dans la taille des FIFOs. Ces derniers sont pernicieux, car le retrait d'une simple case d'une unique FIFO peut les faire apparaître.

Le projet *Buffalo*, mené par Alix Munier et dans lequel Emmanuelle Encrenaz et moi-même sommes impliqués, a pour objectif d'étudier formellement ce problème. On cherche à définir des algorithmes exacts et approchés pour minimiser globalement le nombre d'éléments mémorisants requis par les FIFOs tout en garantissant l'absence d'inter-blocages. Une première étude théorique qui vise à poser le problème de manière formelle a fait l'objet d'une publication dans une conférence nationale [EMP02].

Techniquement, la spécification parallèle est écrite à l'aide de la bibliothèque DPN (*Disydent Process Networks*). Cette bibliothèque réalise les canaux et les primitives de communication. Les tâches sont des *threads* POSIX. Les primitives de communication bloquantes READ() et WRITE() du modèle de Kahn sont implantées en utilisant les moyens de synchronisation standard POSIX. Le comportement des tâches est décrit en langage C. L'utilisation d'un langage très répandu

et d'une bibliothèque standard pour le parallélisme permet d'exécuter cette spécification sur la plupart des systèmes d'exploitation pour station de travail en vue de la validation et de l'évaluation du comportement de l'application. L'utilisation des threads POSIX plutôt que des mécanismes basés sur les processus, comme les pipes Unix, permet un gain très significatif (un ordre de grandeur) en vitesse d'exécution. Expérimentalement, sur une machine mono-processeur, l'implantation parallèle multiplie par deux le temps d'exécution par rapport à la spécification séquentielle.

Un autre gros avantage de cette approche est que les tâches logicielles peuvent être exécutées sur des systèmes d'exploitation embarqués sans modification de la spécification, ni de la bibliothèque de communication.

L'initiative *YAPI* [dKES+00] de Philips possède sa propre implantation au dessus de Pamela, une implantation de *threads* de l'université de Delft [vGe93]. L'intérêt de Pamela est de pouvoir faire de la modélisation de performance sans exécuter le code, mais ce n'est pas notre objectif ici. L'avantage de notre solution est d'être portable sur de nombreux systèmes directement, et notamment de bénéficier des machines multiprocesseurs à mémoire partagées disponibles sur le marché – Sparc Entreprise, Pentium IV, etc – avec des implantations optimisées.

Dans la spécification parallèle, les tâches communiquent entre elles par des canaux. La figure 4.1 illustre les caractéristiques d'un canal.

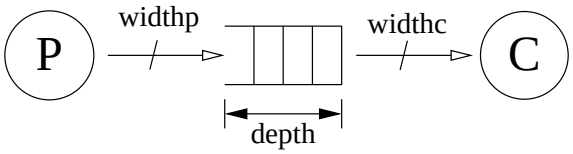

FIG. 4.1 – Un canal de communication entre deux tâches

Un canal contient une fifo dont les caractéristiques sont la profondeur (depth), qui est le nombre de cases dont elle dispose, et la largeur (width) de ses signaux de données en entrée et en sortie. On fait l'hypothèse que width est un multiple de la taille de l'octet.

Une tâche écrit ou lit dans un canal à l'aide de deux primitives bloquantes du point de vue de l'application :

– CHANNELWRITE($channel$, $buffer_p$, $size_p$) ;
– CHANNELREAD($channel$, $buffer_c$, $size_c$) ;

La variable $size$ est la quantité de données que la tâche souhaite lire ou écrire dans un canal. C'est un multiple de width. L'ordre de grandeur de $size$ est très variable (pour une tâche traitant des images par exemple, on peut envisager qu'elle échange un pixel, un bloc de pixels, ou même une image complète). Les tampons $buffer_p$ et $buffer_c$ sont locaux à chaque tâche et ne peuvent être accédés que par leur propriétaire. Les valeurs de $size_p$ et $size_c$ sont indépendantes. De même la profondeur de la FIFO n'est en rien reliée à ces valeurs. La variable $buffer$ est une zone mémoire locale à la tâche suffisamment grande pour contenir $size$ éléments.

La fonction CHANNELWRITE ne retourne à l'application que lorsqu'elle à écrit le nombre de données demandées. Ceci garanti que le tampon $buffer_p$ n'est pas modifié durant l'écriture. Il en est de même pour la lecture bien évidemment.

Cette description du graphe peut être compilée et exécutée sur une station de travail car tous les systèmes d'exploitation actuels implantent les *threads* POSIX. Son exécution permet de valider la spécification parallèle sous forme de graphe de tâches communiquantes par rapport à la spécification séquentielle initiale en comparant leur comportement vis-à-vis des mêmes stimuli d'entrée, comme, par exemple, l'exécution d'une plage mp3 ou d'une séquence vidéo.

DPN permet aussi d'extraire les caractéristiques des communications :
- une analyse de l'exécution de la spécification permet de déterminer la complexité de chaque tâche pour un processeur. Cette analyse est d'autant plus précise qu'elle est réalisée sur le processeur embarqué du système cible ;
- les canaux de communication de DPN permettent de connaître le volume de données échangées entre les tâches et la granularité des échanges si celle-ci est dépendante des données.

4.4 RAFFINEMENTS

La spécification fonctionnelle peut-être annotée avec des informations temporelles, afin de permettre l'évaluation grossière des performances de l'application. Il faut remarquer dès à présent que la seule opération autorisée sur le temps est d'en augmenter la valeur. Il est notamment interdit de le tester pour prendre des décisions qui en dépendent.

D'une part, cette annotation est forcément imprécise, car elle représente la vision qu'a le concepteur de la complexité des calculs. Pour affiner la précision, on peut imaginer que ces informations soient obtenues grâce à un comportement plus bas niveau, comme le fait l'outil *VCC* de Cadence [SK02]. D'autre part, cette simulation prenant en compte le temps est basée sur les mêmes principes que la simulation sans le temps, c'est-à-dire que nous ne sommes pas maître de l'ordonnanceur, car c'est celui des *threads* POSIX qui est utilisé, et qu'il ne possède pas la notion de temps. Ceci ne remet donc pas en cause les propriétés des réseaux de Kahn, et le modèle général de la simulation reste strictement identique au modèle sans temps. On ne cherchera donc pas à homogénéiser les dates des tâches à chaque instant, mais on se contentera de temps locaux à chaque tâche.

Le problème devient alors un problème de mise à jour des horloges locales en fonction des dates des données qui transitent. C'est un problème assez classique en informatique répartie dont la solution est donnée dans un article de Lamport [Lam79]. On peut la résumer ainsi : chaque tâche possède un temps local qu'elle ne peut que faire croître. Chaque donnée produite par une tâche est étiquetée avec la date locale de la tâche productrice. Lorsqu'une tâche consomme une donnée, elle prend la date de la donnée qu'elle consomme si cette date est supérieure à sa date locale, sinon elle conserve sa date locale. Ainsi, de proche en proche et par l'intermédiaire des données les relations de causalité temporelles sont respectées. Cela ne veut pas dire que toutes les tâches ont avancé de manière identique du point de vue de leurs dates locales. En effet, s'il n'existe pas de chemin reliant deux tâches dans le graphe (le graphe n'est pas connexe), alors on ne peut rien dire de leur dates relatives. En pratique, si toutes les tâches concourent à une même application, il existera des chemins entre elles et l'on pourra donc comparer leurs temps de façon significative. Comme l'a remarqué Jean-Lou Desbarbieux, on peut affiner la vision du temps en associant également un temps aux cases vides d'un canal. La lecture a alors pour effet de donner à la première case vide la date de la tâche qui consomme. Ainsi, lors d'une écriture, la tâche productrice voit sa date courante prendre la date de la case de FIFO dans laquelle elle écrit si elle est supérieure à la date locale. Ceci rend strictement symétrique les actions de lecture et d'écriture.

L'ajout du temps ne modifie en rien les propriétés des réseaux de Kahn, car les temps ne sont que des étiquettes qui n'influent en rien sur la production et la consommation des données. Alain Greiner a prouvé que le rapport entre la différence des temps locaux à chaque tâche par rapport au temps de n'importe quelle autre tâche peut être aussi petit que l'on veut en simulant assez longtemps.

L'ajout du temps dans les processus de Kahn nous a été demandé pour la modélisation d'une

interface réseau, dénommée ANI, développée au LIP6 par Franck Wajsbürt et Jean-Lou Desbarbieux.

4.5 EXTENSIONS ET PERSPECTIVES

Le modèle des réseaux de Kahn est très utile pour modéliser la fonctionnalité et les communications d'une application. Cependant, il ne permet pas d'exprimer l'ensemble de ce que l'on voudrait exprimer :

- des contraintes de temps. Les applications temps-réel nécessitent la spécification d'une période et d'une date butoir qu'il est intéressant de pouvoir spécifier. Il est possible de spécifier ces temps, mais à l'heure actuelle le simulateur laisse à l'utilisateur le soin de les vérifier. Le choix d'implantation de la spécification fonctionnelle à l'aide des primitives POSIX ne se prête pas bien à ces vérifications, car comme nous l'avons dit précédemment, le temps global n'existe pas ;
- la gestion d'évènements impromptus. Les applications embarquées interagissent avec l'environnement. Certaines actions extérieures sont rares et ne sont donc pas des données au sens du traitement. Leur occurrence et la date de celle ci étant inconnue – par exemple l'action d'un utilisateur sur une commande à distance – entraîne un non-déterminisme de l'application.
  Une manière de procéder s'intégrant bien dans le modèle conceptuel de Kahn – mais qui bien évidemment ne conserve pas ses propriétés – est de définir une primitive CHANNELSELECT($channel0$, $channel1$, $channel2$, ..., $channelN$). Cette primitive suspend la tâche jusqu'à ce que l'un des canaux présents dans la liste des arguments possède une donnée s'il est en lecture ou une case vide s'il est en écriture. Elle retourne un masque donnant le ou les canaux ayant amené à son réveil. Si la lecture ou l'écriture d'un élément a lieu dans un canal, il y a une probabilité assez forte pour qu'en fait un nombre significatif de telles actions aient été effectuées. Ainsi il n'est pas nécessaire de donner un nombre d'éléments présents dans le canal pour justifier le réveil. Cela permet de simplifier l'implantation, car en aucun cas la le nombre de cases du canal n'est relié au nombre d'éléments que l'on pourrait attendre. Hors le CHANNELSELECT ne consomme ni ne produit de données, donc il faudrait savoir le nombre d'éléments que les producteurs et les consommateurs à l'autre bout des canaux se sont engagés à produire ou à consommer pour revenir de cet appel.
  L'introduction du CHANNELSELECT dans les échanges de données change assez radicalement la sémantique de simulation en introduisant du non-déterminisme. Le comportement du système est alors dépendant de l'ordre d'évaluation des tâches. Pour palier ce problème – lorsque c'est un problème, c.-à-d. que le CHANNELSELECT est utilisé pour modéliser des accès concurrent à une même ressource –, on peut utiliser les annotations de temps pour ordonner les requêtes. Cela rend plus complexe l'exécution du système, car lors de l'exécution, il faut garantir des contraintes de précédences temporelles entre les canaux en attente. Sans l'avoir démontrer formellement, nous avons l'intuition qu'une gestion approprié des temps peut rendre de nouveau la simulation indépendante de l'ordre d'évaluation des tâches. Cependant, cela suppose que ce temps soit respecté lors des phases d'implantation de l'architecture, ce qui en soit est une gageure.
- la création/destruction dynamique de tâches. Les applications de télévision numériques doivent supporter la visualisation simultanée de plusieurs canaux, l'enregistrement en temps réel sur disque, ... L'utilisation optimale des ressources nécessite qu'elles soient gérées dynamiquement, y compris l'exécution de tâches spécifiques.

L'utilisation des *threads* POSIX permet la création dynamique de tâches. En conséquence, il est possible de simuler fonctionnellement des applications qui en font usage sur l'implantation actuelle de DPN.

Même si des pistes apparaissent dans les points soulevés ci-dessus, leur étude poussé reste à faire.

4.6 PROTOTYPES LOGICIELS

Denis Hommais et moi-même avons développé une bibliothèque de communication implantant les deux primitives des réseaux de Kahn ainsi que le CHANNELSELECT à l'aide des *threads* POSIX. Cette bibliothèque a été modifiée par Jean-Lou Desbarbieux pour prendre en compte correctement le temps lors du CHANNELSELECT, et permettre sa distribution sur plusieurs machines.

Plusieurs applications ont été modélisées avec cette technique, et notamment un décodeur de flux d'images au format jpeg et des parties d'un processeur réseau.

4.7 BIBLIOGRAPHIE

[Cal93] Jean-Paul Calvez. *Embedded Real-Time Systems.* John Wiley & Sons, 1993.

[dKES+00] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers. YAPI : Application modeling for signal processing systems. In *Proceedings of the 37th Design Automation Conference*, pp. 402–405. IEEE, June 2000.

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, pp. 471–475, Stockolm, Sweden, August 1974.

[KM77] Gilles Kahn and David B. MacQueen. Coroutines and networks of parallel processes. In *Information Processing*, pp. 993–998, 1977.

[Lam79] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs, In *IEEE Transactions on Computers*, vol C-28, pp. 690–691, September 1979.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Par95] Thomas M. Parks. *Bounded Scheduling of Process Networks.* PhD thesis, Electronics Research Laboratory, Berkeley, 1995.

[vGe93] Arjan J.C. van Gemund. Performance Prediction of Parallel Processing Systems : The Pamela Methodology, in *Proc. ACM International Conf. on Supercomputing*, Tokoi, ACM, July 1993, pp. 318-327.

[BHL+94] J. T. Buck, S. Ha, E. A. Lee and D. G. Messerschmitt. Ptolemy : A framework for simulating and prototyping heterogeneous systems. in International Journal of Computer Simulation, 4 :155-182, April 1994.

[SK02] Frank Schirrmeister and San Krolikoski. The System-level HW/SW Co-design Challenge. Cadence and Partners Craft State-of-the-Art Virtual Component HW/SW Co-Design Software. `http ://www.cadence.com/whitepapers/vcc.html` .

4.8 PUBLICATIONS RELATIVES AU CHAPITRE 4

Conférences internationales avec comité de lecture

[BWLP99] Jean-Yves Brunel, Yosinori Watanabe, Luciano Lavagno, and Frédéric Pétrot. COSY : levels of interfaces for modules used to create a video system on chip. In Jean-Yves Roger, Brian Stanford-Smith, and Paul Kidd, editors, *EMMSEC'99, Buissiness and work in the information society : new technologies and application*, pp. 772–778, Stockholm, Sweden, June 1999. IOS Press.

[BKKP+00] J-Y. Brunel, W. M. Kruijtzer, H. J. H. N. Kenter, F. Pétrot, L. Pasquier, E. A. de Kock, and W. J. M. Smits. COSY communication IP's. In *37th Design Automation Conference*, pp. 406–409, Los Angeles, CA, June 2000.

Conférences nationales avec comité de lecture

[EMP02] Emanuelle Encrenaz, Alix Munier et Frédéric Pétrot. Minimisation de la taille des buffers entre tâches communicantes pour la conception de composants embarqués, Quatrièmes journées nationales de la roadef, Paris, février 2002.

4.9 ENCADREMENT DE THÈSES

[Hom01] Denis Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC soutenue en septembre 2001.

4.10 ENCADREMENT DE STAGES DE DEA

[Buc01] Richard Buchmann. *Interfaçage VCC/CASS/DPN.* Mémoire de DEA ASIME, 2001.

[Pet01] Boriana Petrova. *Modélisation d'un décodeur MJEPG sous VCC.* Mémoire de DEA ASIME, 2001.

Chapitre 5

Simulation « au niveau cycle » de systèmes digitaux synchrones

5.1 Introduction

Ce chapitre présente mon travail dans le domaine de la simulation cycle par cycle des systèmes intégrés digitaux synchrones.

Dans un premier temps, l'objectif est de simuler de la façon la plus efficace possible un système pour lequel on décrit le comportement de chaque composant matériel cycle par cycle. Dans ce cadre, l'exécution des composantes logicielles de l'application n'est rien d'autre que l'exécution d'un modèle de processeur interprétant le code binaire. On utilisera donc des techniques de pré-compilation, ainsi que des modèles exécutables efficacement.

Dans un second temps, nous relâchons la contrainte de précision de sorte à permettre de simuler fonctionnellement des systèmes dont l'architecture de certains composants matériels est encore à définir. Cette simulation permet en effet de déterminer des bornes en latence et débit, qui seront ensuite utilisées comme contraintes lors de la synthèse de ces coprocesseurs.

L'idée de faire de la simulation en utilisant des modèles s'exécutant cycle après cycle n'est pas nouvelle [Jen91]. C'est en fait une extension naturelle de celle retenue par les simulateurs de jeux d'instructions de processeurs, les *Instruction Set Simulators* utilisé lors du développement conjoint de processeurs et de compilateurs. Néanmoins, lorsque j'ai abordé ce sujet en 1996, l'application de ces méthodes à des architectures intégrant des caches, de la mémoire et des périphériques plutôt qu'à un unique composant était encore balbutiante [KVdV97].

5.2 Problématique

Après avoir défini sa spécification parallèle, le concepteur doit faire le choix d'assigner une ou plusieurs tâches sur les éléments constitutifs de l'architecture cible – processeurs programmables ou coprocesseurs câblés – et définir les méthodes de communication entre ces tâches.

Cette phase nécessite d'être capable d'évaluer rapidement le coût des choix, car les approches simplistes ne marchent généralement pas.

En effet, on pourrait croire que faire le profil d'exécution d'une tâche permet de définir la nature matérielle ou logicielle de son implantation. Mais cette approche naïve ignore trois facteurs importants :

1. la loi d'Amdahl [Amd67] qui nous rappelle que le gain en performance que l'on peut espérer en parallélisant est limitée par la séquentialité de l'application ;

2. le coût des communications entre deux tâches de nature différentes. Les transferts de données et la synchronisation qui doivent être mis en œuvre dans ce cas coûtent en bande passante, en latence, et en temps CPU ;
3. des facteurs moins formels mais tout aussi importants de la conception, comme par exemple la possibilité de reprogrammer une partie de l'application, ce qui permet de couvrir une gamme de produits avec un matériel identique. De même, la correction d'erreurs – qu'elles soient dues à une spécification imprécise ou à un problème de conception – est possible *a posteriori* sur du logiciel, alors qu'il est nécessaire de refabriquer le matériel. Le coût – exhorbitant, autour de 500,000 $ en 2002 – d'un jeu de masque rend pertinent une plus grande utilisation du logiciel.

Pour aider le concepteur dans ces choix d'architecture et également lors de la validation fonctionnelle de son application une fois les choix effectués, le niveau « cycle » possède la précision requise. Cependant cette précision à un coût en vitesse d'exécution.

Notre but est maintenant de présenter l'approche que j'ai définie avec Denis Hommais pour minimiser ce coût.

5.3 APPROCHE RETENUE

L'idée initiale est d'utiliser le fait que du point de vue de ces entrées/sorties, le comportement d'un composant au niveau système est synchrone à une horloge, du moins pour la grande majorité de ses ports.

Nous considérons donc le système matériel comme un ensemble de machines à états synchrones qui communiquent.

Il est nécessaire de pouvoir supporter différents domaines d'horloge, car beaucoup de systèmes actuels sont bâtis autour de la notion de « localement synchrone, globalement asynchrone », ce qui en pratique se traduit par la présence de différents domaines d'horloges sur le circuit. Dans de tels systèmes, un module est sensible à une ou plusieurs horloges. On fait l'hypothèse qu'il existe un rapport rationnel entre les fréquences des différentes horloges. Cette hypothèse est nécessaire, car on doit, lors de la compilation et non au cours de l'exécution, définir l'ordre d'évaluation des composants.

Définition d'une machine à états

Une machine à états est définie par le quintuplet $\{I, O, S, t, g\}$, dans lequel :

- I est l'ensemble des valeurs possibles des entrées de la machine à états. On note **I** le vecteur de bits représentant l'encodage de I,
- O est l'ensemble des valeurs possibles des sorties. **O** est le vecteur de bits représentant l'encodage de O,
- S est l'ensemble des états possible de la machine. **S** est le vecteur de bits représentant l'encodage de S,
- t est la fonction de transition. Elle calcule l'état suivant en fonction de l'état courant et des entrées de la machine. t est telle que $t : I \times S \rightarrow S$. **T** est un vecteur de fonctions booléennes calculant **S** en fonction de **I** et de **S**,
- g est la fonction de génération. Elle calcule l'ensemble des sorties de la machine en fonction de l'état courant et de l'ensemble des entrées. g est tel que $g : I \times S \rightarrow O$. **G** est un vecteur de fonctions booléennes calculant **O** en fonction de **I** et de **S**.

Deux types de machines à états sont communément différenciés : les machines de Moore dont la fonction de génération ne dépend que de l'état courant, et les machines de Mealy dont

la fonction de génération dépend de l'état courant et des entrées. Cette distinction est fondamentale dans notre approche.

Définition des fonctions

On note $\mathbf{I}_0$ le vecteur de bits représentant l'ensemble des entrées primaires du système. Les équations (5.1) et (5.2) donnent les propriétés d'une machine de Moore et de Mealy dans le système. Nous noterons $\underset{ck}{\Leftarrow}$ l'affectation sur le front actif de l'horloge. L'indice t indique le cycle d'horloge.

$$\text{Machines de Moore} = \begin{cases} \mathbf{S}^{t+1} & \underset{ck}{\Leftarrow} & \mathbf{T}(\mathbf{I}^t, \mathbf{S}^t) \\ \mathbf{O}^t & = & \mathbf{G}(\mathbf{S}^t) \end{cases} \tag{5.1}$$

$$\text{Machines de Mealy} = \begin{cases} \mathbf{S}^{t+1} & \underset{ck}{\Leftarrow} & \mathbf{T}(\mathbf{I}^t, \mathbf{S}^t) \\ \mathbf{O}^t & = & \mathbf{G}(\mathbf{I}^t, \mathbf{S}^t) \end{cases} \tag{5.2}$$

Pour les machines de Mealy, $\mathbf{O}$ peut être séparée en deux vecteur de bits : $\mathbf{OS}$, le vecteur de bits des sorties ne dépendant pas des entrées et $\mathbf{OC}$, le vecteur de bits des sorties dépendant combinatoirement des entrées. On aura :

$$\begin{cases} \mathbf{OS}^t = \mathbf{GS}(\mathbf{S}^t) \\ \mathbf{OC}^t = \mathbf{GC}(\mathbf{S}^t, \mathbf{I}^t) \\ \mathbf{S}^{t+1} = \mathbf{T}(\mathbf{S}^t, \mathbf{I}^t) \end{cases} \tag{5.3}$$

Où $\mathbf{GS}$ et $\mathbf{GC}$ sont deux vecteurs de fonctions booléennes représentant respectivement la partie de $\mathbf{G}$ calculant le vecteur de bits des sorties ne dépendant pas combinatoirement des entrées $\mathbf{OS}$ et la partie de $\mathbf{G}$ calculant le vecteur de bits des sorties dépendant combinatoirement des entrées $\mathbf{OC}$.

Avec cette notation, une machine de Moore est une machine de Mealy dont l'ensemble des sorties combinatoires est vide : $OC = \varnothing$. Cette séparation permet de dissocier pour une machine à états deux fonctions : une fonction « séquentielle » comprenant la fonction de transition et la fonction de génération de Moore, et une fonction « combinatoire » qui est la fonction de génération de Mealy. La figure 5.1 illustre cette décomposition.

$$\{\mathbf{S}^{t+1}, \mathbf{OS}^{t+1}\} = \mathbf{Seq}(\mathbf{I}^t, \mathbf{S}^t) = \{\mathbf{T}(\mathbf{I}^t, \mathbf{S}^t), \mathbf{GS} \circ \mathbf{T}(\mathbf{I}^t, \mathbf{S}^t)\} \tag{5.4}$$

$$\mathbf{OC}^t = \mathbf{Comb}(\mathbf{I}^t, \mathbf{S}^t) = \mathbf{GC}(\mathbf{I}^t, \mathbf{S}^t) \tag{5.5}$$

5.4 ÉVALUATION D'UN SYSTÈME

L'évaluation d'une machine à états consiste à exécuter d'abord la fonction séquentielle et ensuite la fonction combinatoire.

Évaluation d'un système ne comportant que des automates de Moore

Comme le montre l'équation (5.4), les fonctions séquentielles ne dépendent que des valeurs des signaux et des états calculés au cycle précédent.

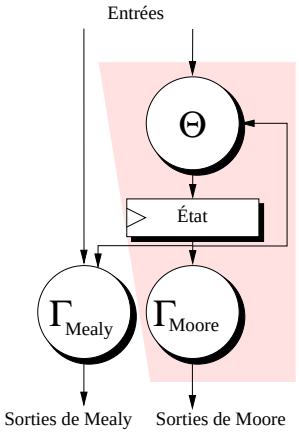

FIG. 5.1 – Décomposition d'un automate pour la simulation cycle

L'évaluation d'une fonction séquentielle ne dépend donc pas de l'évaluation d'une autre fonction séquentielle. L'évaluation des fonctions séquentielles peut donc être faite dans n'importe quel ordre, puisqu'il n'y a pas de dépendances entre les résultats de ces fonctions.

La simulation du parallélisme est obtenue en dissociant la valeur courante de la valeur future des signaux de sorte que la modification d'un signal par une machine ne modifie pas instantanément la valeur en entrée d'une autre machine. Les signaux sont mis à jour après l'évaluation de toutes les fonctions séquentielles. Un système ne comportant que des machines de Moore pourra donc être évalué par l'algorithme de la figure 5.2.

```
MOORESIMULATION()
1  do
2     for i ← 1 to n
3     do O_i ← SÉQUENTIELLE(I_i, S_i)
4     I ← O
5  while simulate
```

FIG. 5.2 – Boucle de simulation d'un système composé uniquement de machines de Moore

La ligne 4 copie la valeur future des signaux dans la valeur courante après l'exécution de toutes les fonctions séquentielles.

Cette modélisation a été présentée lors d'une conférence internationale [PHG97a].

Évaluation d'un système comportant aussi des automates de Mealy

Ici, les signaux de Mealy, c'est à dire les signaux $\mathbf{OC}_i$ introduisent des dépendances entre les fonctions combinatoires. Ces dépendances induisent implicitement un ordre d'exécution optimal, puisque si la partie combinatoire du module m_j possède une entrée issue de la partie combinatoire du module m_i, alors évaluer ces parties dans l'ordre m_i puis m_j exécute une simulation correcte. Un problème se pose si l'on à des dépendances « croisées » entre parties combinatoires de modules.

Pour identifier et résoudre ce problème, nous construisons un graphe dirigé. Les nœuds sont les modules du système qui possèdent des signaux de Mealy (c'est-à-dire les modules qui possèdent une fonction combinatoire non vide). Il existe un arc entre deux nœuds i et j lorsque le module m_j est attaqué par un signal de Mealy issu du module m_i.

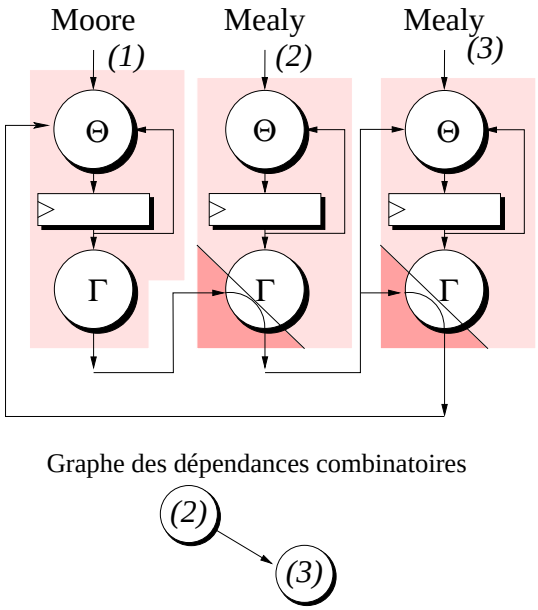

FIG. 5.3 – Construction d'un graphe acyclique

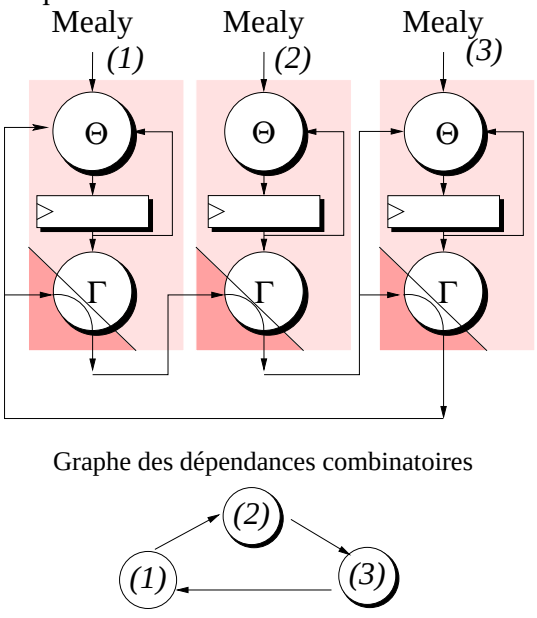

FIG. 5.4 – Construction d'un graphe avec un cycle

La figure 5.4 donne deux exemples simple, l'un possédant un cycle, l'autre non.

Ce graphe peut être composé de plusieurs sous-graphes indépendants dites « composantes connexes ». Ces composantes connexes peuvent être traitées indépendamment et l'ordre d'évaluation entre composantes n'a pas d'importance puisque par définition, elles ne partagent pas d'arcs.

Nous avons déjà identifié que si le graphe est acyclique, alors les fonctions de Mealy peuvent être ordonnées statiquement. Rechercher cet ordre équivaut à étiqueter les nœuds du graphe par un tri topologique. Le tri topologique consiste à numéroter les nœuds de telle sorte que pour deux nœuds u et v s'il existe un chemin de u vers v alors u aura un numéro plus petit que v [CLR94]. L'ordre obtenu en triant les nœuds par étiquette croissante est celui avec lequel les fonctions combinatoires doivent être appelées.

Si le graphe contient des cycles, alors on peut construire un ordre d'évaluation qui, sans être optimal, sera en pratique meilleur qu'un ordre quelconque.

Une présentation plus complète de ces résultats a été publiée dans deux conférences internationales [PHG97b, HP98]. Une ébauche de preuve qu'une telle simulation simule effectivement ce qui est demandé y est donnée.

J'ai également présenté ces travaux lors de deux colloques internationaux [Pét99b, Pét99a] liés à des projets européens.

Ces travaux ont aboutis à l'implantation du simulateur CASS, abréviation de *Cycle Accurate System Simulation*.

5.5 SIMULATION DE COPROCESSEURS MATÉRIELS AVANT SYNTHÈSE

Denis Hommais et moi avons défini une méthode de co-simulation permettant d'exécuter un sous-système précis au cycle et des tâches DPN destinées à la synthèse matérielle. La simulation d'un système dont les coprocesseurs ne sont pas synthétisés est un programme multi-*threads* comprenant autant de *threads* qu'il y a de tâches réalisées par des coprocesseurs matériels et un *thread* qui est le simulateur CASS et qui lance les autres *threads* par des modules adaptateurs. La figure 5.5 montre un système dans lequel il y a deux *threads*, T_2 et CASS.

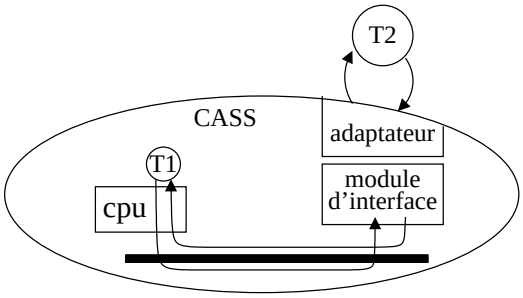

FIG. 5.5 – *threads* exécutées

5.6 SIMULATION DE MODÈLES VHDL

Les techniques de simulation au cycle près peuvent être appliquées également à la simulation de modèles VHDL décrits au niveau RTL ou post synthèse, après abstraction fonctionnelle. En effet, les équations constituant ces modèles ne comportent pas de cycles car les registres du circuit sont donnés. En conséquence, on peut séquentialiser son comportement parallèle entre

deux registres en construisant un graphe dirigé acyclique que l'on étiquette ensuite grâce à un tri topologique. On peut alors construire, pour chaque registre, une fonction de transition et une fonction de génération.

J'ai développée un prototype de traduction de VHDL flot de données repris et grandement amélioré par François Donnet et Christophe Alexandre. J'ai effectué avec Ludovic Jacomme le même travail pour un sous-ensemble de VHDL comportemental. Ce travail a été supporté par le projet MEDEA AT 403.

5.7 PERFORMANCES DE CASS ET COMPARAISON AVEC SYSTEMC

Les techniques mises en œuvre dans CASS visent toutes à accélérer la simulation, en utilisant des étapes de pré-compilation dès que cela est possible. Ceci permet d'atteindre des vitesses de simulation de l'ordre du mégahertz pour des systèmes simples sur des machines à 1GHz. On peut tomber à 50Khz pour des systèmes possédant un très grand nombre de composants, et bien évidemment, plus il y a de composants et plus ils sont complexes, plus la vitesse de simulation est lente.

Depuis deux ans, l'initiative SystemC menée par Synopsys, vise à remplacer le VHDL pour la modélisation du comportement de composants au niveau système. Dans la continuité des efforts fait par Synopsys pour la simulation VHDL et Verilog, le cœur du simulateur est événementiel. Un modèle SystemC est donc comparable, du point de vue de ses entrées/sorties à un modèle VHDL.

Afin de faire une comparaison objective entre les temps de simulation obtenus par CASS et ceux obtenus par SystemC, nous devons :

- utiliser exactement les mêmes modèles dans les 2 simulateurs. Pour cela, nous devons tout d'abord redéfinir les primitives CASS d'accès aux signaux en primitives SystemC. Ceci ne pose pas de problèmes sémantique car l'écriture d'une valeur n'a lieu qu'après le Δ-cycle d'évaluation des modèles dans les 2 environnements. Les modèles CASS écrits en C doivent par contre être encapsulés pour être utilisé sous SystemC qui requiert du C++. Cette transformation est automatisable ;
- contraindre l'échéancier assurant la gestion des évènements sous SystemC à se comporter comme un simulateur au niveau cycle lorsque cela est possible. Nous devons alors décomposer chaque cycle en trois étapes :
  1. exécution des fonctions combinatoires jusqu'à stabilité ;
  2. exécution unique de chaque fonction séquentielle ;
  3. copie des signaux dans les registres (barrière de synchronisation).

Il suffit d'établir une liste de sensibilité particulière pour obtenir ce type de fonctionnement. Nous choisissons arbitrairement :

- au front descendant :
  - Exécution des fonctions *Combinatoires* jusqu'à stabilité ;

  La prise en compte du front descendant permet d'assurer que chacune des fonctions *Combinatoires* sera évaluée sans faire apparaître explicitement les sorties des registres d'états sur les interfaces. C'est une condition *sine qua none* pour permettre l'utilisation directe des modèles CASS sous SystemC, car ces états sont symboliques et n'apparaissent donc pas sur les interfaces. Cela permet une meilleure abstraction et également une grande factorisation des évènements et donc un gain en performance ;
- au front montant :
  - exécution unique de chaque fonction *Séquentielle* ;

– copie des signaux dans les registres.

Avec SystemC, nous obtenons ainsi une évaluation unique des fonctions séquentielles et un ordonnancement dynamique de nos fonctions combinatoires initialement prévues pour la simulation par relaxation. Cette technique selon nous est aussi équitable que possible pour la comparaison de performances.

Richard Buchmann a simulé différents programmes sur une architecture composée d'un processeur, de 4 bancs mémoire et d'un contrôleur de bus, et obtenu les valeurs présentées dans la table 5.1.

Système/Simulateur	CASS	SystemC 2.0	CASS/SystemC 2.0
PGCD	356.693 cycles/s	50.633 cycles/s	7,0
LEE2	1.090.068 cycle/s	139.664 cycles/s	7,8
MEMTEST	1.160.000 cycle/s	136.000 cycles/s	8,5

TAB. 5.1 – Comparaison des performances de CASS et de SystemC.

Les performances sont, au minimum, de l'ordre d'un facteur 7 en faveur de CASS. Ceci est dû à l'échéancier, aux propagations d'évènements rendues inutiles, et aux temps d'accès aux ports d'entrées/sorties. Le gain supplémentaire induit par le seul ordonnancement, est plus difficile à apprécier car dépendant des implémentations. Les écarts de performance augmentent avec l'importance de l'ordonnancement statique.

Ce travail est essentiellement du à Richard Buchmann, dont le premier objectif est de permettre à CASS d'accepter sans modifications des modèles SystemC décrit avec le formalisme précédent. Ce travail a fait l'objet d'une publication dans une conférence nationale [BPG02].

Le second objectif que j'ai fixé à Richard Buchmann est d'étudier la transformation d'un modèle SystemC peu contraint dans le formalisme de CASS, pour positionner CASS comme un simulateur rapide de SystemC. Ces objectifs vont de paire avec l'action spécifique bibliothèque d'IPs pour la simulation (SOCLIB) portée par le CNRS dans le cadre du réseau thématique SoC.

5.8 ÉVALUATION DE LA CONSOMMATION AU NIVEAU CYCLE

Le problème de la consommation dans les systèmes sur puce est majeur. Ana Abril, doctorante ciffre, s'est attaquée à ce problème en prenant comme exemple un décodeur vidéo MPEG en cours de développement chez Philips. L'objectif de son travail est de définir une méthode d'estimation au niveau cycle pour les parties matérielles et logicielles. L'objectif n'est pas ici de quantifier la consommation en valeur absolue, mais de définir des métriques permettant de comparer plusieurs implantations en elles. Pour cela, l'équipe d'encadrement d'Ana Abril, – Jean Gobert et Thomas Dombek (Philips Research France), ainsi qu'Habib Mehrez et moi-même (LIP6) – l'a poussée à définir des équations de consommation instantanée d'énergie, l'énergie étant ensuite sommée cycle après cycle pour donner une puissance consommée.

Le modèle retenu, puisque la simulation se fait cycle par cycle, et d'associer une consommation d'énergie à une transition dans l'automate d'état. Un tel modèle peut être défini *a priori* ou extrait *a posteriori* d'une implantation existante [Dup02]. Des premiers résultats ont été obtenus et publiés dans deux ateliers internationaux [Abr02].

5.9 CONCLUSION ET PERSPECTIVES

Selon nous, la méthode de modélisation utilisée dans CASS est optimale pour le niveau bit précis et cycle précis. L'implantation du simulateur lui même est proche de l'optimale, puisque

l'évaluation des parties combinatoire est généralement unitaire sauf quand une telle partie appartient à un cycle. L'unique optimisation restant à effectuer, mais dont nous ne sommes pas sur qu'elle aura un impact en pratique, concerne la liste de sensibilité. En effet, la liste de sensibilité est une information grossière dans le sens ou elle n'indique pas quel signal d'entrée influence quels signaux de sortie. Si cette information était disponible, le graphe des dépendances de données pourrait être affiné, et le nombre de cycles du graphe diminué, car on éliminerait ainsi des « faux » cycles dus au manque de précision de l'information.

À court terme, nous nous intéressons à être capable de simuler des systèmes décrits avec un sous ensemble bien identifié de SystemC à l'aide de CASS. Cette stratégie permettra de mettre en évidence les qualités de notre approche pour la simulation cycle tout en permettant aux concepteurs de rentrer dans le moule d'un langage standard de modélisation.

5.10 PROTOTYPES LOGICIELS

Denis Hommais et moi-même avons développé le simulateur CASS, pour *Cycle Accurate System Simulation*, ainsi que des modèles de composants utilisable autour du PI-Bus, par exemple un processeur R3000 avec ses caches, une mémoire capable de lire du format exécutable `ecoff`, un arbitre de bus, etc. Nous avons également spécifié et implanté le module matériel de support à la synthèse des communications décrit au chapitre 6. Denis Hommais a également développé un extracteur de stimuli de simulation permettant de récupérer des vecteurs de test utilisables pour simuler une implantation VHDL de la même spécification cycle.

Richard Buchmann a développé les mécanismes permettant d'encapsuler les modèles CASS pour les rendre utilisables sous SystemC. Ses objectifs sont aussi de permettre à CASS de prendre directement en entrée des modèles SystemC de différents niveaux.

Denis Hommais et Pascal Gomez ont développé l'interface de simulation entre le monde précis au cycle de CASS et des tâches externes s'exécutant sur l'hôte du simulateur.

Ivan Augé a développé une sortie CASS pour l'outil de synthèse de haut niveau UGH, ce qui permet de simuler au cycle près des coprocesseurs synthétisés. L'approche ici ne nécessite pas de passer par un outil de synthèse logique, et la boucle synthèse de haut niveau suivie de simulation est très rapide – de l'ordre de 5 minutes – pour des circuits d'une dizaine de milliers de portes. La vitesse de simulation est très proches de celle obtenue avec des modèles écrits par le concepteur, puisque les opérateurs sont conservés lors de la synthèse de haut niveau.

Ludovic Jacomme, François Donnet, Christophe Alexandre et moi même avons développé des outils permettant de traduire des sous-ensembles du VHDL en modèles CASS utilisant des primitives définies par Franck Wajsbürt.

5.11 COLLABORATIONS AVEC L'INDUSTRIE

Le développement de l'outil CASS ainsi que celui des différents traducteurs de VHDL vers des modèles de simulation CASS a été supporté par le projet MEDEA AT 403, *System Level Methods and Tools*, de 1997 à 2001.

La thèse d'Ana-Belen Abril Garcia [Abr01] est une thèse ciffre en collaboration avec Philips Research France.

5.12 BIBLIOGRAPHIE

[Amd67] G.M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pp. 483–485, Reston, Va., April 1967. AFIPS Press.

[Chr75] Nicos Christofides. *Graph Theory, An Algorithmic Approach*, chapter 10, Hamiltonian Circuits, Paths and the Traveling Salesman Problem, pp. 214–235. Academic Press, 1975.

[KVdV97] Sjaak Koot, Arjan Versluys, and Pieter de Visser. *TSS User Manual*. Philips Semiconductors, September 1997.

[CLR94] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Graph Algorithms*, chapter 23, pp. 485–488. MIT Presss, 2nd printing, 1994.

[RF80] D. F. Robinson and L. R. Foulds. *Digraph structure*, chapter 2, pp. 43–62. Gordon and Breach Scientific Publishers, 1980.

[Jen91] G. Jennings. A case against event driven simulation for digital system design. In *24th Annual Simulation Symposium*, pp. 170–175, April 1991.

5.13 PUBLICATIONS RELATIVES AU CHAPITRE 5

Conférences internationales avec comité de programme

[PHG97a] Frédéric Pétrot, Denis Hommais, and Alain Greiner. A simulation environment for core based embedded systems. In *Proceeding of the* 30^{th} *IEEE International Simulation Symposium*, pp. 86–91, Atlanta, Georgia, April 1997. IEEE.

[PHG97b] Frédéric Pétrot, Denis Hommais, and Alain Greiner. Cycle precise core based hardware/software system simulation with predictable event propagation. In *Proceeding of the* 23^{rd} *Euromicro Conference*, pp. 182–187, Budapest, Hungary, September 1997. IEEE.

[HP98] Denis Hommais and Frédéric Pétrot. Efficient combinational loops handling for cycle precise simulation of system on a chip. In *24th Euromicro*, pp. 51–54, Vesteras, Sweden, August 1998. IEEE.

[Abr02] Ana-Belén Abril Garcia, Jean Gobert, Thomas Dombek, Habib Mehrez et Frédéric Pétrot. Energy Estimations in High Level Cycle-Accurate Descriptions of Embedded Systems In *The 5th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 228-235, Brno, Czech Republic, April 2002.

Ateliers internationaux

[Pét99a] Frédéric Pétrot. Cycle accurate system simulation. In *Medea Workshop on Cosimulation*, Eindhoven, The Netherlands, May 1999.

[Pét99b] Frédéric Pétrot. Cycle accurate system simulation. In *Medea-Esprit Conference*, Antwerpen, Belgium, September 1999.

Conférences nationales avec comité de programme

[HPG97] Denis Hommais, Frédéric Pétrot, et Alain Greiner. Un environnement de simulation pour les systèmes embarqués. *Actes du premier colloque CAO de circuits et systèmes*, pp. 66–69, Villard de Lans, France, Janvier 1997.

[BPG02] Richard Buchmann, Frédéric Pétrot, et Alain Greiner. Ordonnancement statique versus pilotage évènementiel. *Actes du troisième colloque CAO de circuits et systèmes*, pp. 151–154, Paris, France, Mai 2002.

5.14 ENCADREMENT DE STAGE DE DEA

[Dup02] Yoan Dupret. *Modélisation de la consommation dans des descriptions précises au cycle.* Mémoire de DEA ASIME, juin 2002. En co-encadrement avec Ana Abril et Habib Mehrez.

5.15 ENCADREMENT DE THÈSES

[Hom01] Denis Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC, 13 septembre 2001.

[Buc01] Richard Buchmann. *Simulation rapide de systèmes intégrés.* Thèse en cours.

[Abr01] Ana Abril. *Estimation et optimisation de la consommation dans les systèmes intégrés haute performance.* Thèse ciffre avec Philips Digital System Labs à Paris, Jean Gobert (PDSL), Thomas Dombek(PDSL), Habib Mehrez(LIP6) et moi-même.

Chapitre 6

Synthèse des communications et support matériel/logiciel

6.1 Introduction

La synthèse des communications consiste à réaliser physiquement les canaux permettant les échanges de données entre les tâches, qu'elles soient logicielles et/ou matérielles. L'assignation d'une tâche sur un processeur ou sur un accélérateur matériel est déterminée par le concepteur. Ainsi, la synthèse des communications doit être rapide à mettre en œuvre pour permettre plusieurs itérations afin d'explorer rapidement plusieurs alternatives d'implantation du système.

L'exploration architecturale consiste tout d'abord à effectuer les deux actions décrites sur la figure 6.1 – c.-à-d. décider de la nature logicielle ou matérielle d'une tâche et choisir l'implantation effective de la communication – puis à simuler le système résultant pour en déduire ses performances.

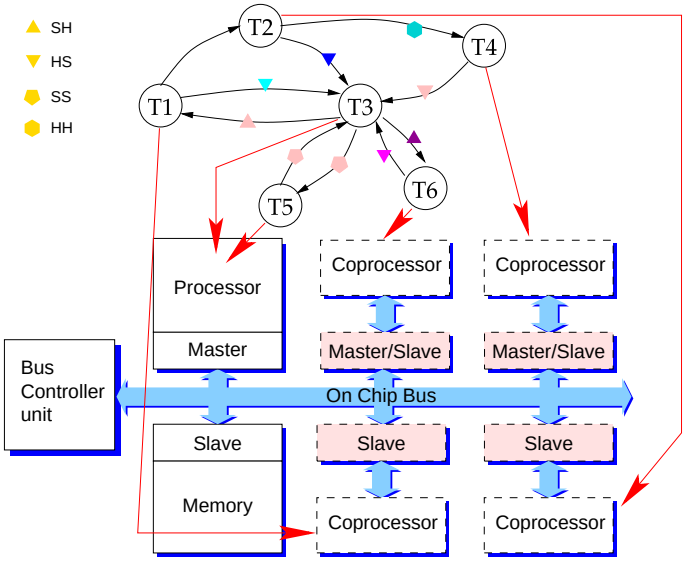

FIG. 6.1 – Affectation des tâches et des modes de communications

Nous avons opté pour des communications tout à fait spécifiques, par l'intermédiaire de canaux à l'exclusion de toute autre, comme nous l'avons vu au chapitre 4.

Ce travail a été mené par Denis Hommais et moi-même au LIP6. Il est inspiré des travaux de nos collègues du laboratoire de recherche de Philips à Eindhoven, en particulier Jean-Yves Brunel et Wido Kruijtzer, avec lesquels nous avons travaillé sur ce sujet dans le cadre du projet Esprit COSY. Nous avons participé à un article présenté dans une conférence internationale avec eux sur ce sujet [BKK+00].

6.2 APPROCHE GÉNÉRALE

L'idée est de prédéfinir un certain nombre de « schémas de communication » instanciés en fonction de la nature matérielle ou logicielle des tâches productrice et consommatrice, et de la quantité et du débit des données à échanger.

Il ne s'agit pas à proprement parler d'un « outil de synthèse » mais d'« instanciation » de composants matériels et logiciels prédéfinis. Cette instanciation est décidée par le concepteur, en fonction des besoins de l'application. Il faut donc simplement associer à chaque canal du graphe de tâches un schéma de communication particulier.

En l'occurrence, cette instanciation repose sur l'existence de primitives implantant les fonctions CHANNELREAD et CHANNELWRITE, soit en logiciel, soit en matériel.

Ces travaux ont été publiés dans deux conférences internationales [HPA01a, HPA01b].

Implantation des communications

Communications logiciel/logiciel

À titre d'exemple, nous présentons l'implantation de la communication entre deux tâches logicielles effectuée par le processeur. Ces primitives utilisent deux services proposés par les *threads* POSIX : le `mutex` pour l'accès aux données partagées et la `condition` pour la suspension des tâches sur les entrées–sorties. Les figures 6.2 et 6.3 donnent l'algorithme de l'écriture et de la lecture du canal. On notera $data$ le tampon du canal, $status$ le nombre de cases pleines dans ce tampon, $write$ son pointeur d'écriture et $read$ son pointeur de lecture.

Sur ce pseudo-code, on remarque que seul le registre $status$ est protégé par un verrou car c'est la seule ressource de la fifo qui peut être modifiée par le consommateur et le producteur. Les pointeurs de lecture et d'écriture ne sont accédés et modifiés que par une seule tâche (respectivement le consommateur et le producteur). Les données sont protégées par le registre $status$: le consommateur ne lit pas plus de données qu'il y en a dans la fifo d'après le registre $status$. De même, le producteur n'écrit pas plus de données qu'il n'y a de cases vides dans la fifo d'après ce même registre. Ceci permet de limiter la prise du `mutex` à des temps très courts, et permet donc de bénéficier au mieux du parallélisme vrai s'il existe.

On voit sur ce code que les tâches peuvent être activées et suspendues plusieurs fois, en fonction de la taille de la donnée à échanger et de la profondeur de la FIFO. Néanmoins, ceci est caché par les fonctions de lecture/écriture, et c'est cela qui permet de rendre indépendant la taille des échanges et le nombre de données dans la FIFO.

Une autre manière évidente de faire communiquer deux tâches logicielles consiste à programmer un DMA qui va effectuer le transfert à la place du processeur. Cependant cette façon de faire n'est pas efficace pour des petites quantités de données, car le DMA doit informer les tâches de la fin de son transfert par interruption.

```
CHANNELWRITE(channel, buf, size)
 1  n ← size
 2  forever
 3  do p ← depth − status
 4     m ← min(n, p)
 5     if m ≤ depth − write
 6       then MEMCPY(data + write, buf, m)
 7       else end ← depth − write
 8            MEMCPY(data + write, buf, end)
 9            MEMCPY(data + write, buf + end, m − end)
10     buf ← buf + m
11     write ← (write + m)  mod depth
12     n ← n − m
13     MUTEX_LOCK(channel)
14     status ← status + m
15     COND_SIGNAL(channel)
16     if n = 0
17       then break
18     if depth − status = 0
19       then COND_WAIT(channel)
20     MUTEX_UNLOCK(channel)
21  MUTEX_UNLOCK(channel)
22  return size
```

3 calcul de la place disponible

4 m vaut soit le nombre de données à copier s'il y a suffisamment de place et la place disponible sinon

5–10 copie de m données dans le tampon circulaire et mise à jour du pointeur d'écriture

13–14 mise à jour de l'état de la fifo protégée par un verrou

15 réveille la tâche attendant éventuellement l'écriture de la fifo

16–17 si tout a été copié, sortie de la boucle

19 sinon attente de la lecture de la fifo

21 relâchement du verrou

FIG. 6.2 – Écriture dans un canal en utilisant les services des *threads* POSIX

Autres communications

Les communications matériel/matériel ou matériel/logiciel font forcément intervenir du matériel spécialisé. Dans le chapitre 4 de sa thèse [Hom01], Denis Hommais présente un petit nombre d'implantations possibles de communications entre des tâches de diverses nature. Ces implantations permettent de mettre en évidence les ressources que le matériel assurant l'interface doit posséder. Elles n'ont pas de prétention d'exhaustivité, mais on peut espérer, vu la diversité des services qu'elles rendent, que l'invention de nouveaux schémas ne nécessitera pas de ressources nouvelles dans le matériel.

6.3 MODULE MATÉRIEL DE SUPPORT À LA COMMUNICATION

La définition d'un certain nombre de manières d'effectuer la communication entre deux tâches, quelle que soit leur nature, spécifie les ressources matérielles nécessaires à la communication.

Afin de s'abstraire du médium d'interconnexion effectivement utilisé, nous avons opté pour une structure qui suit les niveaux d'abstraction proposés au paragraphe 3.1. Néanmoins, en pratique il faut définir des protocoles au bit près permettant d'implanter la sémantique de ces niveaux, comme illustré sur la figure 6.4. Ainsi, un coprocesseur désirant communiquer grâce aux primitives de lecture/écriture définies au chapitre 4 ne pourra le faire que s'il est connecté à un module spécial implantant les ressources nécessaires au support de la communication. Les données transitant entre le coprocesseur et le module d'interface s'échangent grâce à un protocole dit « vecteur » qui permet, comme les primitives de lecture/écriture, de fournir au système l'information du nombre de données transmises. Une version dégradée de l'interface permet

```
CHANNELREAD(channel, buf, size)
 1  n ← size
 2  forever
 3  do p ← status
 4     m ← min(n, p)
 5     if m ≤ depth − read
 6       then MEMCPY(buf, data + read, m)
 7       else end ← depth − read
 8            MEMCPY(buf, data + read, end)
 9            MEMCPY(buf + end, data + read, m − end)
10     buf ← buf + m
11     read ← (read + m)  mod depth
12     n ← n − m
13     MUTEX_LOCK(channel)
14     status ← status − m
15     COND_SIGNAL(channel)
16     if n = 0
17       then break
18     if status = 0
19       then COND_WAIT(channel)
20     MUTEX_UNLOCK(channel)
21  MUTEX_UNLOCK(channel)
22  return size
```

3 lecture du nombre de données présentes dans la fifo,

5–11 lecture des données du tampon circulaire,

15 signalement de la lecture à la tâche attendant pour écrire d'autres données dans la fifo,

16–17 si *size* données ont été lues ; sortie de la boucle de lecture,

19 sinon, attente de l'écriture de données par une autre tâche,

21 relâchement du verrou.

FIG. 6.3 – Lecture dans un canal en utilisant les services des *threads* POSIX

de fonctionner en mode « fifo », mais alors les schémas de communication qui ont besoin du nombre de données promises lors de la requête ne sont plus utilisables. Cela peut être particulièrement dommageable pour la construction de paquets (*burst*) pour le transfert physique des données sur l'interconnexion.

Le module d'interface doit par ailleurs communiquer avec le reste du monde. L'idée ici est de s'appuyer sur le standard Virtual Component Interface [VCI00] défini par le consortium VCI. C'est grâce à Jean-Yves Brunel de Philips que Denis Hommais et moi avons été impliqués dans le développement de cette initiative, que nous avons suivie dès le début. La figure 6.5 présente l'architecture générale du maître/esclave VCI (que nous nommerons dans la suite *vcims*) connecté à des convertisseurs de protocoles (dénommés ensuite *wrappers*) pour PI-Bus. Le module est paramétrable. Il possède jusqu'à 4 fifos cibles d'entrée et 4 fifos cibles de sortie, et optionnellement une fifo initiatrice en entrée et une fifo initiatrice en sortie. Les limites des paramètres permettent un débit crête d'un mot par cycle à 100 MHz pour une technologie 0.35μm sur un PI-Bus. En terme très pragmatique de plus petit commun dénominateur pour la réutilisation, le choix de VCI est indiscutablement « la » solution. On peut rétorquer que VCI n'est pas aujourd'hui un succès industriel, mais toutes les autres solutions sont propriétaires, et aucune n'a la généricité de VCI. La figure 6.6 illustre la vision que VCI donne d'une architecture pour le développement d'applications embarquées. Cette vision est compatible avec les plates-formes présentées sur la figure 3.3 page 38.

Comparé à une implantation purement PI-Bus, le passage par le protocole VCI en mode esclave coûte un cycle de latence supplémentaire lors de l'accès à des ressources différentes et rien en débit, même pour des paquets de longueur indéterminée, ce qui est acceptable. Le coût en mode maître est nul. Nous possédons ces données car, dans le cadre du projet COSY, nous

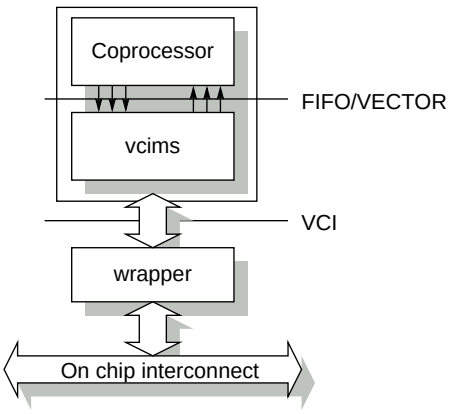

FIG. 6.4 – Niveaux d'abstraction visibles lors de l'implantation.

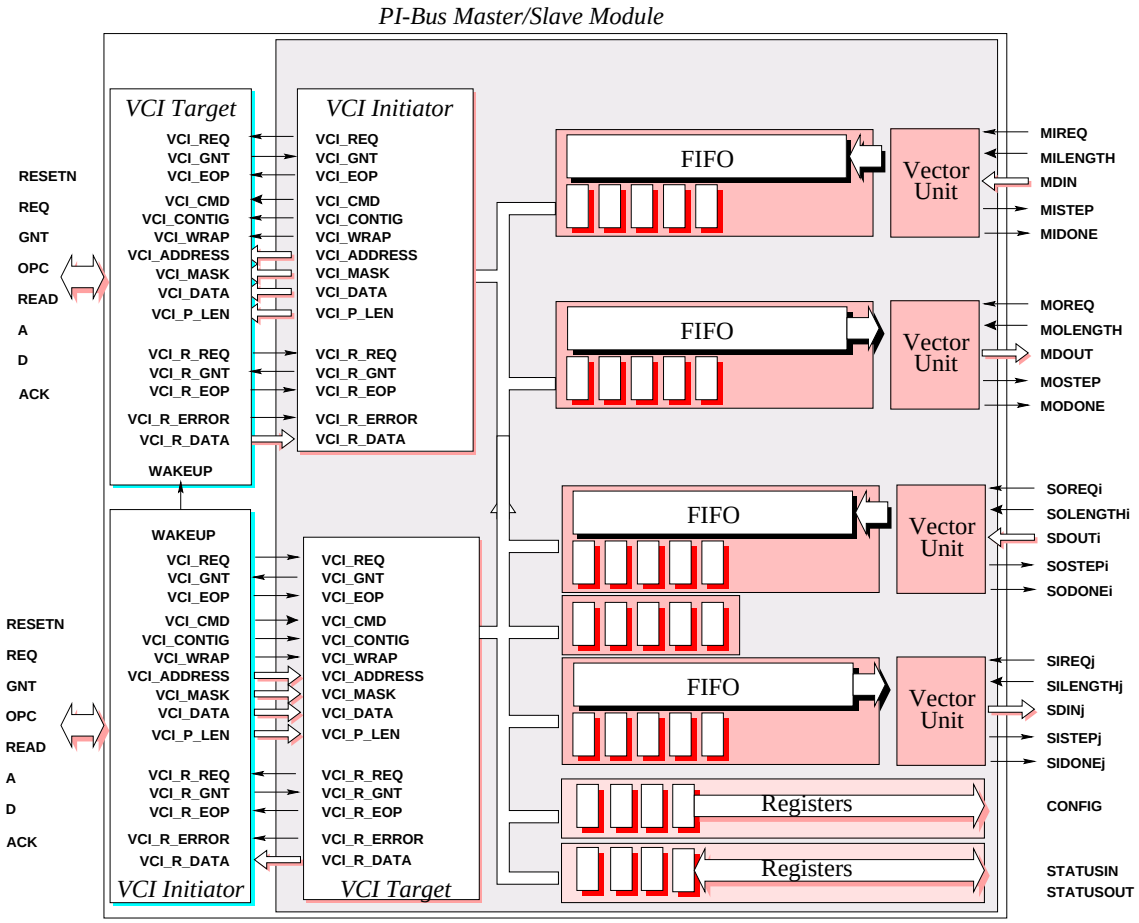

FIG. 6.5 – Vue d'ensemble du module d'interface.

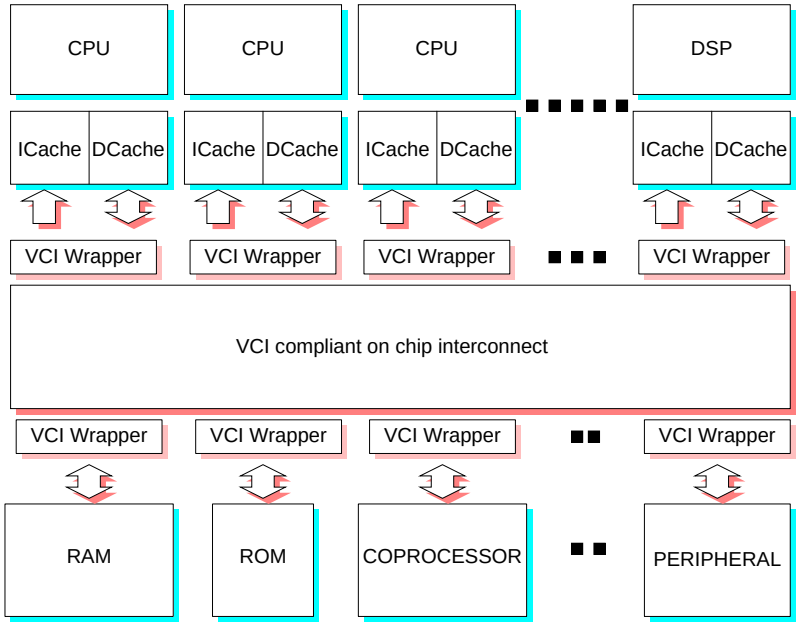

FIG. 6.6 – Vision VCI d'une plate-forme.

avons commencé par implanter nos modules directement sur une interface PI-Bus que nous avons ensuite refaite pour passer à VCI.

Le fait d'utiliser ce standard garantit que l'on peut transférer physiquement des données. Il ne garantit en rien que la sémantique des communications est assurée. C'est donc l'utilisation que l'on en fait grâce à des pilotes logiciels ou à d'autres modules matériels que l'on doit le respect de cette sémantique : en effet rien en théorie ne s'oppose par exemple à ce qu'une primitive logicielle soit non-bloquante. Il suffit qu'au lieu de suspendre la tâche lors de l'accès à une ressource non disponible, la fonction retourne à l'appelant.

Ce module est présenté, pour ce qui concerne sont aspect conceptuel, dans la thèse de Denis Hommais [Hom01]. Une présentation sous forme de *datasheet* est donnée dans [PH00a, PH00b] pour les concepteurs désireux de l'utiliser en pratique.

6.4 PROTOTYPES LOGICIELS ET MATÉRIELS

Denis Hommais et moi-même avons développé la partie logicielle d'un certain nombre de schémas de communication. Nous avons également défini l'architecture du module matériel de communication (le *vcims*), là encore en collaboration avec Jean-Yves Brunel et Wido Kruijtzer de Philips Reseach. C'est Jean-Yves Brunel qui nous a incité à étudier l'interface VCI avant même que la première version du standard soit rendue publique.

Denis Hommais a développé le modèle CASS du *vcims*, ainsi que des *wrappers* maîtres et esclaves pour le PI-Bus. Alain Houelle et Nicolas Vaucher en ont fait l'implantation VDHL synthétisable optimisée.

6.5 COLLABORATIONS AVEC L'INDUSTRIE

Le développement des modèles de simulation cycle et VHDL synthétisable du *vcims* et des *wrappers* pour PI-Bus a été supporté par le projet ESPRIT COSY, *Codesing Simulation and Syntesis*, de 1997 à 2000. Les modèles de simulation ont été portés sous TSS, le simulateur cycle de Philips, et utilisé par Philips dans le démonstrateur de COSY.

6.6 BIBLIOGRAPHIE

[VCI00] On-Chip Bus Development Working Group. VSI Alliance Virtual Component Interface Standard. On-Chip Bus 2.2.0, août 2000.

6.7 PUBLICATIONS RELATIVES AU CHAPITRE 6

Conférences internationales avec comité de lecture

[HPA01a] Denis Hommais, Frédéric Pétrot, and Ivan Augé. A practical toolbox for system level communication synthesis. In *9th International Symposium on Hardware/Software Co-design*, pages 48–53, Copenhagen, Dennemark, April 2001.

[HPA01b] Denis Hommais, Frédéric Pétrot, and Ivan Augé. A toolbox to map system level communications on hw/sw architectures. In *12th International Workshop on Rapid System Prototyping*, pages 77–82, June 2001.

[BKK+00] J-Y. Brunel, W. M. Kruijtzer, H. J. H. N. Kenter, F. Pétrot, L. Pasquier, E. A. de Kock, and W. J. M. Smits. COSY communication IP's. In *37th Design Automation Conference*, pages Pages 406–409, Los Angeles, CA, June 2000.

Manuels

[PH00a] Frédéric Pétrot and Denis Hommais. Rapport COSY D4.1.1, 55 pages, Novembre 2000.

[PH00b] Frédéric Pétrot and Denis Hommais. Rapport COSY D4.2.1, 15 pages, Novembre 2000.

6.8 ENCADREMENT DE THÈSES

[Hom01] Denis Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC soutenue en septembre 2001.

6.9 ENCADREMENT DE STAGES DE DEA

[Big97] Jean-François Bignolle. *Modules de communications génériques pour PI-Bus.* Mémoire de DEA ASIME, 1997. Avec Denis Hommais.

[Pas98] Laurent Pasquier. *Modélisation cycle de modules de communications.* Mémoire de DEA ASIME, 1998. Avec Denis Hommais.

[Gom99] Pascal Gomez. *MJEPG : Conception d'un système intégré, de la spécification fonctionnelle au modèle de performance cycle true* Mémoire de DEA ASIME, 1999. Avec Denis Hommais.

[Fau02] Etienne Faure. *Exploration et synthèse architecturale d'un décodeur MJPEG.* Mémoire de DEA ASIME, 2002. Avec François Donnet.

Chapitre 7

Synthèse d'architecture

7.1 Introduction

La synthèse d'architecture consiste à produire une description au niveau transfert de registres à partir de la description d'un algorithme. On sent bien que le nombre de manières différentes dont peut-être effectuée la synthèse est gigantesque, et que ce processus est, bien plus encore que la compilation de programmes, influencé par le style de l'écriture de la spécification d'entrée. L'approche généralement adoptée en synthèse d'architecture est constituée de trois phases[1] :

Allocation
: l'allocation consiste à choisir pour chaque opération du comportement un opérateur appartenant à une bibliothèque de macro-cellules connue de l'outil de synthèse. Par exemple, une addition peut être effectuée sur un simple additionneur, un additionneur/soustracteur, ou une boîte à opérations ;

Binding
: cette phase consiste à associer une instance d'un opérateur particulier parmi toutes les instances de ce modèle dans le chemin de données à une opération effectuée dans le comportement. En d'autres termes, on choisit lors de cette phase le chemin d'exécution d'une expression du comportement ;

Scheduling
: l'ordonnancement consiste à fournir à chaque cycle de fonctionnement la liste des opérations à effectuer. C'est cette séquentialisation des opérations qui de fait va assurer que le circuit se comporte conformément à sa spécification.

Les travaux en synthèse d'architecture au sein de notre groupe sont menés par Ivan Augé. L'idée défendue par Ivan Augé est qu'envisager la synthèse d'architecture sous la forme *Allocation*, *Binding*, *Scheduling* est un cercle vicieux. La meilleur preuve étant qu'alors que cette discipline a plus de 20 ans d'existence, et qu'en théorie elle apporte un gain très important en productivité pour le concepteur d'un coprocesseur, aucun outil commercial généraliste n'a réussi à percer réellement à ce jour. Briser ce cercle consiste à demander au concepteur de guider la synthèse. Une première approche serait d'indiquer dans le comportement, à l'aide de pragma, les *desiderata* d'implantation pour telle ou telle expression. Une seconde consiste à exprimer directement une machine à état en terme de macro-états, et de dire à chacun de ces états quels sont les ressources utilisées [Sug00]. On se rapproche alors d'une conception plus classique, en laissant alors l'outil de synthèse RTL effectuer le partage des opérateurs, en exigeant alors une

spécification cycle par cycle et non comportementale.

Une troisième solution, celle retenue par l'équipe et implantée dans l'outil Ugh, consiste à demander au concepteur de fournir explicitement, en plus de son comportement, un chemin de données simplifié sur lequel on exécutera le comportement. Cette approche est assez naturelle si l'on considère que la conception de circuit est une affaire de concepteur. Ainsi, l'étude d'architecture qu'il faut mener dans tous les cas lors de la conception d'un circuit fera toujours ressortir un chemin de données. L'idée est de permettre au concepteur d'exprimer simplement ce chemin de données, puis de l'utiliser pour la réalisation de son comportement.

7.2 APPROCHE GÉNÉRALE

Le concepteur décrit son comportement dans un dialecte du langage C, dialecte qui permet la compilation par un compilateur classique à fin de simulation par exemple. Ce dialecte est nécessaire essentiellement pour spécifier le nombre de bit de chaque variable du programme, et restreint l'utilisation de certaines constructions non synthétisables [Don02]. Il décrit ensuite son chemin de données de manière simplifiée sous forme d'un graphe de dépendances entre registres et opérateurs. Ni les entrées de données, ni les commandes de multiplexeurs, ni les drapeaux nécessaires aux prises de décisions dans l'automate ne sont explicités. Cette description est raffinée en un chemin de données précis au bit près en recherchant la manière d'effectuer une expression sur le graphe de dépendance, tout en faisant les choix d'implantation. Un article publié dans une conférence internationale présente cette approche dans sa généralité [ABG+97]. Comme on peut le constater, la présence d'un chemin de données simplifié permet de définir l'allocation des registres et partiellement l'allocation et le binding des opérateurs. Le résultat est que l'essentiel de l'effort consiste alors à effectuer l'ordonnancement. Les détails de cette approche ont été publiés dans une conférence internationale [ABGP99].

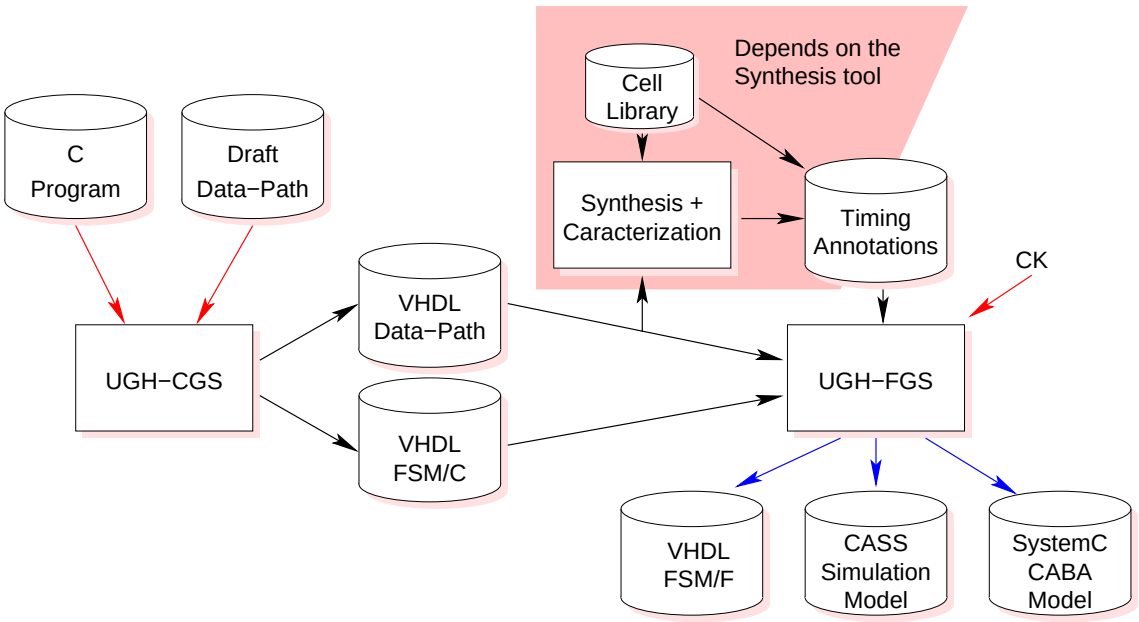

FIG. 7.1 – Présentation générale de la méthode Ugh

La figure 7.1 résume les différentes étapes de l'outil Ugh, dont le nom signifie *User Guided High Level Synthesis*.

La première phase, l'ordonnancement à gros grain dite *Coarse Grain Scheduling*, prend en

entrée un comportement C et un chemin de données simplifié. En utilisant les relations exprimées dans le chemin de données simplifié, l'outil cherche à construire les chemins d'exécution de chaque expression. Ces chemins ne sont que partiellement fournis, et plusieurs chemins peuvent être acceptables. L'outil fait alors des choix afin de minimiser le nombre de multiplexeurs et le nombre de bits de chaque opérateur. La recherche des chemins elle-même est rendue non-triviale par la présence d'unités fonctionnelles effectuant de multiples opérations et par les propriétés mathématiques des opérateurs du langage de spécification. Le résultat de cette phase est un chemin de données complètement spécifié, au bit près, au niveau RTL synthétisable, et une machine à états dans laquelle chaque état est un « pas de contrôle » indépendant de la durée de la ou des opérations contrôlées.

L'étape suivante consiste à synthéthiser chaque opérateur, puis placer et router le chemin de données. La synthèse opérateur par opérateur est sous optimale, mais elle a le mérite de simplifier l'implantation. En effet, les phases suivantes vont utiliser des temps exprimés entre les ports d'entrée et les ports de sortie des opérateurs, et donc la recherche de ces ports est grandement facilitée si l'on opère hiérarchiquement. Notons néanmoins que théoriquement rien dans la méthode ne s'oppose à une synthèse à plat. Le circuit obtenu est ensuite extrait, et les temps de propagation entre opérateurs sont insérés dans la vue structurelle entre unités fonctionnelles.

L'étape d'ordonnancement à grain fin, *Fine Grain Scheduling*, va permettre de restructurer chaque « pas de contrôle » de la machine à états en en séparant et/ou fusionnant les éléments. Pour faire cela, cette phase a besoin du temps de cycle du circuit à synthétiser. Il s'agit alors d'utiliser cette donnée pour faire les choix de fusion/séparation en utilisant les temps fournis dans le chemin de données. On peut alors être amené à faire durer une opération ou une suite d'opérations combinatoires sur plusieurs cycles. Pour se faire, on se contente de ne pas autoriser les écritures dans les registres sources, ou de ne pas permettre la modification de la commande d'un multiplexeur. Ce type d'ordonnancement ne permet pas de savoir à l'avance à quel cycle précis les opérations d'entrées/sorties auront lieu, car ces dates sont des résultats de la synthèse. En pratique, pour les coprocesseurs que nous visons, ce comportement n'est pas gênant. En effet, au niveau système (*cf.* le paragraphe 3.1) les échanges s'effectuent grâce à des fifos qui permettent de tamponner les données aussi bien en entrée qu'en sortie, comme illustré sur la figure 7.2. Le résultat de cette phase est une machine à état qui donne cycle par cycle les

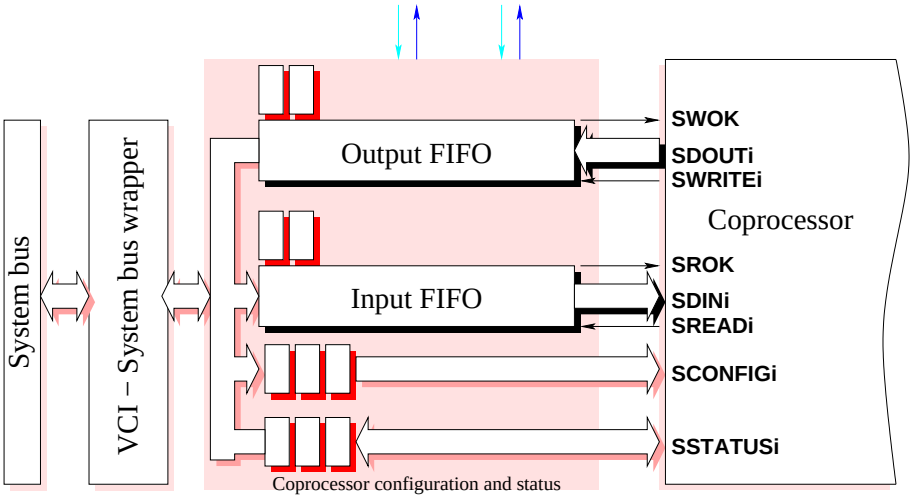

FIG. 7.2 – Connexion d'un coprocesseur à un bus.

commandes à effectuer sur le chemin de données. L'implantation se décline sous la forme de modèle RTL synthétisable ou de modèles C simulables par CASS.

7.3 EXEMPLE D'UTILISATION

Dans le cadre du projet COSY, Ivan Augé et moi avons développé un modèle synthétisable du décodeur de Huffman (VLD) inclus dans la norme MPEG2. La validation du modèle a été effectuée sur le sous-système de la figure 7.3.

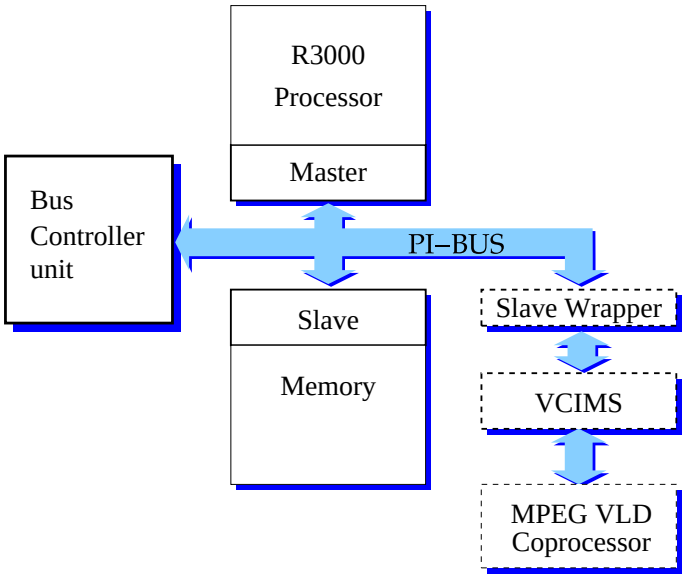

FIG. 7.3 – Environnement de validation du VLD.

Le code fait une centaine de lignes pour ce qui concerne la partie chemin de données simplifiés et autour de 800 lignes de VHDL pour le comportement. La synthèse prend de l'ordre de la demi-heure, y compris la synthèse RTL des opérateurs du chemins de données par *Design Compiler* de Synopsys et l'extraction des temps. Notons que *Behavioral Compiler* de Synopsys échoue par manque de mémoire si on lui fournit le source tel quel, et ce après 4 jours de calculs sur une machine avec 4 GO de mémoire. Si l'on extrait les ROMs du modèle, il aboutit après une dizaine d'heures à un circuit dix fois plus gros que celui issu de notre synthèse. Pour nous, cela signifie que les outils de synthèse doivent permettre de capturer les connaissances du concepteur pour être efficace.

La rapidité de la synthèse et la génération de modèles de simulation précis au cycle en C permet d'obtenir en moins d'une journée des courbes de performance du type de celles présentées sur la figure 7.4. Ces courbes montrent, pour un chemin de données fixe, les variations du séquencement en fonction de la fréquence désirée du circuit.

Ce profile globalement hyperbolique s'explique car les temps d'exécutions sont par définition inversement proportionnels à la fréquence ! Les « décrochements » sont dues au phénomène discret suivant : lors de la réduction du temps de cycle, un chemin entre registres qui pouvait s'effectuer en un cycle ne le peut plus, et il faut donc deux cycles pour le réaliser. Ces deux cycles sont suffisants pour une gamme de fréquences, mais si la fréquence diminue encore, il en faudra trois, et ainsi de suite. Si de tels chemins se trouvent à l'intérieur d'une boucle qui détermine en grande partie le temps d'exécution, comme par exemple dans le décodage bloc dans le cas du MPEG2, alors on obtient le comportement temporelle représenté.

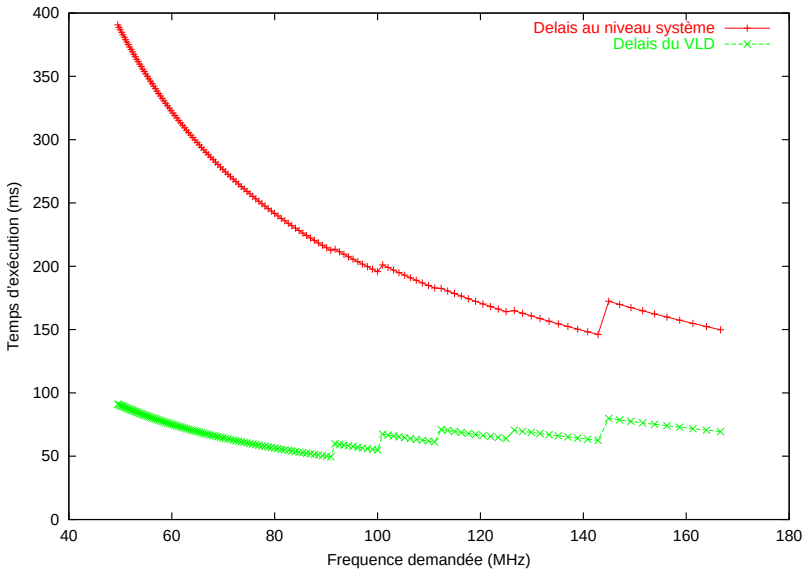

FIG. 7.4 – Temps d'exécution du VLD sur une séquence MPEG2.

7.4 CONCLUSION ET PERSPECTIVES

Dans la méthode de synthèse proposée, le concepteur est mis à contribution et fournit sa vision simplifiée du chemin de données. Ce travail est nécessaire si l'on ne veut pas aboutir à des monstres, comme l'on montré des expérimentations que nous avons menées avec Behavioral Compiler de Synopsys. De plus, c'est le vrai travail du micro-architecte que de définir avec soin son chemin de données, et cela ne représentent en générale pas la partie la plus longue du travail. *A contrario*, la définition cycle par cycle de la machine à état ne nécessite pas de créativité particulière, mais demande de nombreuses itérations avant de converger. Il est donc censé, du point de vue du concepteur de circuits, de faire les choix qui ont été faits pour Ugh. Une retombé de ce choix est que la synthèse d'un circuit, y compris la synthèse RTL, peut être très rapide, car les algorithmes n'ont pas la complexité des algorithmes classiques de compilation et de synthèse de haut niveau.

Cette rapidité est un vrai atout, car ainsi de multiples variations de l'architecture peuvent être essayées. Le couplage avec les outils de simulation cycle et l'adaptation des entrées sorties à la sémantique des lectures/écritures bloquantes permet d'utiliser directement la synthèse des communication pour la mise en œuvre.

Certains algorithmes de Ugh peuvent être améliorés – comme par exemple la prise en compte des propriétés des opérateurs –, et des extensions – le déroulement de boucles, etc. – peuvent être envisagées. Ce travail est en cours [Don00], là encore sous la direction d'Ivan Augé.

7.5 PROTOTYPES LOGICIELS

Ugh est constitué en pratique de nombreux programmes effectuant chacun une étape du processus de synthèse. L'architecture générale de l'outil et les structures de données utilisées ont été principalement définies par Ivan Augé, Ludovic Jacomme, Alain Greiner, François Donnet et moi. La première version de l'ordonnancement à gros grain à été développée par Rajesh Bawa (post-doc ASIME), Pierre Guerrier et Nicolas Pronine (étudiants du DEA ASIME). Nous avons repris cette partie ensuite, avec Stéphane Levassort (étudiant du DEA ASIME), notamment pour générer des modèles de simulation VHDL intermédiaires. Finalement, Francois Don-

net (doctorant) a entièrement repris cette étape, après avoir effectué l'entrée en langage C en utilisant le front-end de GCC modifié par Ludovic Jacomme (post-doc ASIME).

La génération des scripts de synthèse pour Synopsys et l'extraction des temps a été réalisé par un élève de l'Institut d'Informatique d'Entreprise du CNAM à Évry, sous la direction d'Ivan Augé et avec ma participation active.

Ivan Augé, avec l'aide de 4 étudiants de l'IIE et trois DEA ASIME à mené à bien l'implantation de l'ordonnancement à grain fin et la génération des modèles pour la synthèse RTL et la simulation cycle sous CASS.

7.6 COLLABORATIONS AVEC L'INDUSTRIE

Le développement d'un prototype de l'outil Ugh avec une entrée VHDL a été supporté par le projet ESPRIT COSY, *Codesing Simulation and Syntesis*, de 1997 à 2000.

7.7 BIBLIOGRAPHIE

[Aug90] Ivan Augé. *Une méthode de synthèse d'architecture* Thèse de l'Université D'Evry, décembre 1990.

[Bru96] Jean-Yves Brunel. *Synthèse d'architecture de circuits intégrés : Étude de l'ordonnancement au niveau transfert de registres pour prendre en compte les contraintes temporelles imposées par les bibliothèques d'opérateurs disponibles.* Thèse de l'Université Pierre et Marie Curie, décembre 1996.

[1] Daniel D. Gajski, Nikil Dutt, Allen Wu, and Steve Lin. *High Level Synthesis, Introduction to Chip and System Design*, chapter 7 and 8, pp. 213–296 Kuwer Academic Publisher, 1992.

[Sug00] Zoltán Sugár. *Synthèse comportementale basée sur l'ordonnancement.* Thèse de l'Institut National Polytechnique de Grenoble, mai 2000.

[Don02] François Donnet. Représentation des opérateurs du langage C pour la synthèse haut-niveau, *Actes du troisième colloque CAO de circuits et systèmes*, pp. 3–6, Paris, France, Mai 2002.

7.8 PUBLICATIONS RELATIVES AU CHAPITRE 7

Conférences internationales avec comité de lecture

[ABG+97] Ivan Augé, Rajesh K. Bawa, Pierre Guerrier, Alain Greiner, Ludovic Jacomme, and Frédéric Pétrot. User guided high level synthesis. In Ricardo Reis and Luc Claensen, editors, *VLSI : Integrated Systems on Silicon*, pp. 464–475, Gramado, Brazil, August 1997. IFIP, Chapman & Hall.

[ABGP99] Ivan Augé, Jean-Yves Brunel, Alain Greiner, and Frédéric Pétrot. Fine grain scheduling. In *25th Euromicro*, Vesteras, Sweden, September 1999. IEEE.

Manuels

[AP00a] Ivan Augé et Frédéric Pétrot. Document COSY D6.4.1, 87 pages.

[AP00b] Ivan Augé et Frédéric Pétrot. Document COSY D5.4.1, 14 pages.

7.9 ENCADREMENT DE THÈSES

[Don01] François Donnet. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC en cours. Participation à l'encadrement de cette thèse dont la direction est assurée par Ivan Augé.

7.10 ENCADREMENT DE STAGES DE DEA

[Lev99] Stéphane Levassort. *Ordonnancement à gros grain dans UGH.* Mémoire de DEA ASIME, 1999. Avec Pierre Guerrier.

[Fau02] Etienne Faure. *Exploration et synthèse architecturale d'un décodeur MJPEG.* Mémoire de DEA ASIME, 2002. Avec François Donnet.

[Fal02] Steve Falempin. *Extraction et comparaison de chemins d'exécution.* Mémoire de DEA ASIME, 2002. Avec François Donnet.

Chapitre 8

Noyau de système d'exploitation distribué pour multiprocesseur embarqué

8.1 INTRODUCTION

Comme nous l'avons vu dans la problématique générale, les spécifications initiales des applications intégrées sont naturellement parallèles. De plus, lorsque les performances/consommation/surface attendues sont acceptables, le concepteur fera toujours le choix d'une implantation logicielle car cela permet des corrections de bugs après fabrication et supporte des changements très tardifs de la spécification, ce qui n'est pas rare en pratique. Autre avantage, un même **SoC** peut être utilisé pour une gamme de produits. S'il est programmable, il est spécialisable *a posteriori*, et permet donc de toucher un plus large public à peu de frais. Il apparaît donc évident que le logiciel embarqué est une composante essentielle du système intégré. L'évolution naturelle des **SoCs** va amener les architectes à concevoir de véritables ordinateurs parallèles intégrés [LvM+98, GG00]. Cette tendance est affirmée par exemple dans le projet MEDEA A502 MESA au niveau européen [Mes01]. Dans ces architectures, les processeurs peuvent être de même type (homogène) ou de types différents (hétérogène). Je m'intéresse ici à des multiprocesseur homogènes. Je vais donc préciser les aspects de la conception d'un micro-noyau multi-tâches pour multiprocesseur dans la suite de ce chapitre.

8.2 PROBLÉMATIQUE

Partant du constat que les spécifications sont multi-tâches et les plateformes multiprocesseur, il faut que le logiciel puisse s'exécuter comme un ensemble de tâches sur la plateforme, de manière identique à la spécification initiale s'exécutant sur la station de travail. D'autre part, comme certaines tâches sont implantées comme des coprocesseurs matériels, il faut assurer la synchronisation et l'échange de données entre logiciel et matériel. Enfin, il faut en général permettre la prise en compte de contraintes temps réel, telle des latences minimales ou des dates d'exécutions de certaines tâches.

Les problèmes principaux liés aux noyaux pour **SoC** sont :

compacité et performance
: les noyaux disponibles sont généralement organisés sous forme de couches, qui si elles

rendent plus claire l'organisation, nuisent à la compacité du code. De plus, les applications **SoC** sont généralement coopératives et ne nécessitent pas de protection, au sens *user/superviser* des processeurs, et cela permet un gain non négligeable en performance lors de l'accès aux périphériques notamment ;

simplicité, compréhension et maîtrise du code
une bonne compréhension du code, des intéractions avec le matériel, et la possibilité de le modifier pour une application ou une architecture particulière sont obligatoires pour viser des implantations optimisées. De plus, la plupart des noyaux possèdent une *API* propriétaire, ce qui ne permet pas simplement de porter une application, ni de les comparer ;

multiprocesseur
de nombreux noyaux ne s'intéressent qu'à des architectures uniprocesseur. Hors l'intégration, le coût des masques et le besoin de souplesse aidant, il est évident que des machines massivement multiprocesseur vont émerger *on-chip* ;

multiples bancs mémoires
lors de l'utilisation de media d'interconnexion complexes, il n'est pas rare que la mémoire devienne le goulet d'étranglement. La répartition des données doit donc être effectuées, sous contrôle, sur des bancs mémoires accessibles simultanément. Cette possibilité n'est, à ma connaissance, pas prise en compte dans les micro-noyaux existants.

abstraction du matériel
ce problème est présent à différents niveaux : 1- sur le processeur, pour la changement de contexte, masquage des interruptions, ... 2- pour l'accès au périphériques, qui généralement ne possèdent pas d'interfaces homogènes, 3- pour les caches, dont on ne sait pas, par exemple, s'ils sont capables d'assurer la cohérence avec la mémoire, 4- sur les interconnexions, qui participent potentiellement à la cohérence, mais aussi à la consistance de la mémoire, 5- sur les accès atomiques, qui ne peuvent être implantés par une simple rétention du bus dans les réseaux sur puce. Même un vérouillage de la cible peut être problématique à cause des latences d'accès.

L'ambition que je poursuis ici n'est pas de donner une réponse optimale à tout ces problèmes. J'ai chercher à les identifier et à y apporter des solutions suffisamment simples et s'appuyant autant que possible sur du matériel existant. Ceci m'a permis, avec l'aide de Denis Hommais et Pascal Gomez, de mettre sur pied une plateforme experimentale sur laquelle, j'espère, il sera possible de bâtir des solutions originales, mesurables par simulation, et tirant partie des particularités des **SoCs**.

8.3 APPROCHE RETENUE

Permettre une exécution multi-tâches et multiprocesseur nécessite l'existence d'un noyau qui *primo* sait effectuer la commutation de tâches, *secundo* permet la synchronisation entre tâches, et *tertio* gère les interruptions pour les échanges avec le matériel. Dans notre modèle de plate-forme architecturale, la mémoire est partagée par toutes les tâches de l'application. Ceci implique que les processeurs aient tous la même vision de cet espace.

À partir de ces contraintes, mon objectif est de définir un noyau multiprocesseur, multi-tâches et multi-bancs minimaliste. Plutôt que de spécifier des objets avec une sémantique spécifique et une interface de programmation *ad-hoc*, nous avons, parmi les API existantes, opté pour celle proposée par la norme POSIX 1003 qui défini des *threads*. Le choix d'implanter une partie des threads POSIX repose sur le fait que c'est la seule norme – au sens standard accepté et disponible sur un grand nombre de plateformes – actuelle définissant des *threads*. De plus, si les

primitives de communication sont déjà décrites grâce aux threads POSIX au niveau fonctionnel, alors une implantation embarquée purement logicielle ne nécessitera que la recompilation de ces primitives.

L'étude de la spécification des threads POSIX m'a convaincu de sa qualité : l'implantation est en effet légère et fournit un petit nombre de primitives dont la sémantique est bien définie et le comportement justifiable.

On désire d'autre part pouvoir faire évoluer ce noyau en fonction de deux critères :

1. le choix d'implantation du noyau :
   - ordonnanceur unique distribué (SMP : Symmetric Multi-Processors) ou un ordonnanceur global auquel sont adjoints un ordonnanceur par processeur, ou encore d'autres solutions plus statiques ;
   - avec ou sans préemption, ce qui permet de faire du temps partagé ;
   - avec ou sans contraintes temps-réel données aux tâches.
2. l'architecture cible :
   - autour d'un bus unique ce qui permet simplement d'assurer la cohérence mémoire par matériel grâce à la diffusion (*broadcast*) implicite des données et des adresses ;
   - autour d'une hiérarchie de bus ou d'un réseau d'interconnexion. Dans ce cas la diffusion des adresses pour invalidation est extrèmement coûteuse. Les travaux de Censier et Feautrier [CF78] ont permis de minimiser grandement (trois ordres de grandeur) les diffusions par filtrage les informations. La technique proposée requière un répertoire centralisé dont la taille croit linéairement avec la taille de la mémoire et avec le carré du nombre de processeurs. La machine DASH de Stanford [LW95] à 16 groupes (*clusters*) de 4 processeurs utilise néanmoins ce protocol pour la cohérence entre les groupes car il a le mérite de la simplicité. Dans cette lignée, de nombreux travaux ont cherché à réduire l'encombrement mémoire sans trop détériorer les perfromances. Une synthèse relativement récente de ces techniques est fournie au chapitre 4 de [LW95].

Un noyau *SMP* préemptif a été défini et implanté. Il fait l'hypothèse que la cohérence mémoire est assurée par le matériel, ce qui signifie que le moyen d'interconnexion de l'architecture cible est un simple bus et que les caches des processeurs espionnent les lectures sur le bus (*snoop*) pour mettre à jour leur cache de données.

Se pose alors le problème de la cohérence mémoire entre les données manipulées par les différents *threads*, aussi bien pour le système lui-même que pour les couches applicatives ou de communication qui se trouvent au dessus. Le problème ne se pose pas pour les caches d'instructions car on fait l'hypothèse a) que les processeurs ne possèdent pas de mémoire virtuelle ; b) que l'exécution du code ne modifie pas le programme lui même.

L'intérêt des caches n'est plus à démontrer, d'autant que sur des systèmes possédant un grand nombre de composants nécessitant une grande bande passante vont voir la latence des accès mémoire croître fortement (une valeur de 40 à 50 cycles ne sera pas abérante, mais reste de 5 a 10 fois plus rapide que les solutions hors puce telle la SGI 3000). De plus, l'accès à une ligne de cache permet de construire une requête de type rafale (*burst*) qui permet de ne pas payer cette latence pour chaque donnée.

S'il n'est pas possible d'espionner les transferts, alors il faut envisager d'autres solutions. Il faut considérer dans cette hypothèse les deux cas différents déjà énoncés plus haut :

- la cohérence entre les données partagées entre tâches. Pour assurer la cohérence des données partagées, la solution la plus généralement retenue dans les systèmes parallèles est d'utiliser des messages pour échanger des données au lieu de partager une mémoire distribuée. Le problème de la cohérence est alors résolu puisqu'il ne se pose plus !

Il existe cependant des machines massivement multiprocesseur à mémoire partagée, tel DASH [LLG+] dans le monde académique, ou SGI chez les constructeurs. Il est vraisemblable que c'est de cette seconde classe de machine que les **SoCs**, qui peuvent posséder une mémoire interne de taille non négligeable et accessible rapidement, doivent s'inspirer. Si l'on cherche à faire simple, il faut utiliser un tampon d'écriture à écriture systématique (*write through*). Deux solutions simples sont :

1. utilisation de l'espace non-caché pour toutes les données partagées. Le coût risque d'être non négligeable, mais peut-être supportable si les processeurs ont eux-mêmes plusieurs contextes matériels (*simultaneous multithreading*) comme celui de [BMA02] , que la bande passante de l'interconnexion est grande, et que les données partagées sont peu nombreuses ;

2. invalidation sélective des lignes de caches préalablement à la lecture. Cette solution naïve et peu efficace par rapport à l'approche de [CF78] et de ses dérivés, s'implante simplement grâce à une instruction du processeur permettant l'invalidation de tout ou partie du cache.

– la cohérence des données locales à une tâche lorsque les tâches migrent. Là encore deux solutions sont immédiatement possibles :
  – invalidation systématique de la totalité du cache lors d'une commutation de tâches. Ainsi, on garantit que l'on ne retrouvera pas de données issus d'une existence précédente de la même tâche dans le cache. Le coût de cette approche est d'une part le temps d'invalidation du cache, qui en pratique est proportionnel à son nombre de lignes, et le temps nécessaire au rechargement des données. Une optimisation envisageable est de n'invalider le cache que lorsqu'il y a migration effective, ce dont l'ordonnanceur peut s'apercevoir aisément ;
  – une autre piste de travail est l'utilisation de mémoires locales à un processeurs dans lesquels les données sont gérées directement par le programme. C'est une mémoire que l'on peut accéder en 1 ou 2 cycles, comme un cache, mais dont la mise à jour est de la responsabilité du programme lui même. On appelle ce type de mémoire de *scratch-pads*. Leur usage est clairement lié à la possibilité d'avoir un compilateur permettant de les utiliser de manière transparente pour l'utilisateur, ce qui n'est pas le cas aujourd'hui. Les coûts associés à une telle approche ne sont pas clairs car, dans un tel environnement, le contexte d'exécution d'une tâche inclus sa mémoire *scratch-pad*. Cela peut purement et simplement interdire la migration des tâches ou la préemption si l'on fait des hypothèses sur la durée de vie des variables. Bref, l'interaction entre cohérence, compilateur et noyau reste à étudier en détail.

Afin de pouvoir expérimenter autour de ce noyau, nous en avons également fait une version non symétrique dans laquelle chaque tâche est attachée à un processeur. Dans ce cas, un ordonnanceur local est utilisé pour élire une tâche parmi celles s'exécutant sur le processeur particulier. La manipulation des objets POSIX impose une coordination avec les ordonnanceurs distants qui peuvent éventuellement détenir ces objets.

Ces expérimentations, effectuées par Pascal Gomez, devraient permettre de mettre en évidence les avantages – pas d'invalidation systématique des caches – et les inconvénients – pas de répartition de la charge –. Ces résultats sont vraisemblablement très dépendant de l'application et de l'architecture cible.

8.4 IMPLANTATION

Le noyau est classiquement structuré en couches avec, comme couche haute, l'*API* POSIX, et comme couche basse, une *API* spécifique permettant d'accéder les ressources de l'architecture et du processeur. Les processeurs utilisés pour le développement et l'expérimentation sont des MIPS R3000 et l'interconnexion est soit un PI-Bus, soit un réseau SPIN. Le fonctionnement en mode système du noyau et de l'application permet d'éviter les appels systèmes dans tous ces cas et d'avoir un gain significatif en performance. La perte de sécurité engendrée n'est pas critique pour les applications embarquées.

Les fonctions de bas niveau sont écrites soit en C – la grande majorité –, soit en assembleur pour le processeur cible. Le fait que la plupart des fonctions soient écrites en C ne doit pas masquer leur dépendance vis-à-vis de l'architecture au sens large du système, comme par exemple l'accès à un sémaphore ou la manière de gérer des interruptions vectorisées. Les parties dépendantes du processeur sont :

- la commutation de contexte de deux tâches ;
- la partie commune du gestionnaire d'interruption, qui effectue une sauvegarde du contexte, l'appel d'une fonction C de distribution des interruptions et une restitution du contexte ;
- le masquage des interruptions ;
- le code d'initialisation des processeurs ;
- l'invalidation des caches.

Les parties dépendantes de l'architecture sont :

- la prise de sémaphore physique (verrou) ;
- la gestion des interruptions vectorisées ;
- l'invalidation optionnelle des caches ;

La prise d'un sémaphore physique est une opération atomique, il est donc nécessaire de disposer d'un mécanisme matériel de *test and set*. Nous disposons pour cela de sémaphores matériels. Afin d'éliminer les attentes indéterminées, un *thread* ne doit prendre un sémaphore physique que pendant une période de temps bornée et, de préférence, la plus courte possible. Vu que, dans le cas général, il n'est pas possible pour un noyau de détecter les interblocages, il est du ressort du programmeur de s'assurer du bon fonctionnement de son application. Il doit par exemple envisager l'éviction de la tâche courante lors d'une interruption, et soit s'assurer que cela lui convient, soit entrer en section critique avant d'acquérir le sémaphore en n'en sortir qu'après l'avoir relâché. Pour éviter de faire de l'ordonnanceur la ressource la plus contrainte, chaque objet qui le peut possède son propre verrou physique.

L'implantation du noyau multiprocesseur symétrique implique le partage de l'ordonnanceur par tous les processeurs. L'ordonnanceur peut être accédé soit par les *threads* lors d'appels POSIX, soit lors d'une interruption. Nous utilisons un verrou unique pour le partage de l'ordonnanceur. En pratique, ce verrou n'est pris que pour déplacer un *thread* d'une liste à l'autre.

Les manipulations rapides de l'ordonnanceur sont assurées par l'utilisation intensive des listes doublement chaînées dont on possède des pointeurs de tête et de queue. Les listes correspondantes à quatre des cinq états possibles des *threads*, qui sont présentés figure 8.1. Pour l'état `RUN`, on possède une unique entrée par processeur.

Il peut arriver qu'un processeur n'ait plus de *threads* à exécuter. Il se met alors en attente soit d'une interruption matérielle, soit d'une décision de l'ordonnanceur, cette décision étant prise sur un autre processeur.

D'autres listes de *thread* sont présentes dans le noyau. Elles se superposent aux listes de l'ordonnanceur, et lorsque c'est possible, ces éléments possèdent leur propre verrou physique

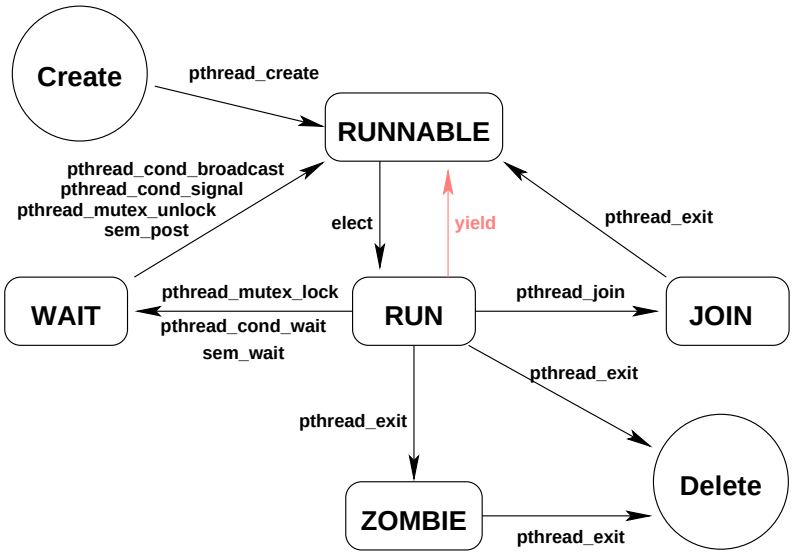

FIG. 8.1 – États possibles d'un *thread*.

afin de minimiser les conflits d'accès à l'ordonnanceur.

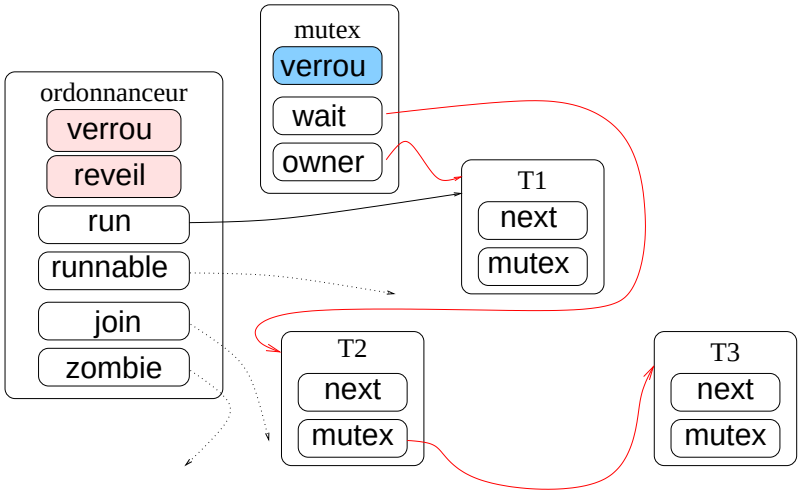

FIG. 8.2 – Structure de l'ordonnanceur.

Comme illustré sur la figure 8.2, chaque *mutex* est associé à un sémaphore physique qui doit être pris par le *thread* qui veut modifier les listes pointées par le *mutex*. Le champs `réveil` permet d'indiquer aux processeurs en attente de traitement qu'une tâche a été réveillée. Il est modifié par toutes les fonctions qui font passer une tâche de l'état `WAIT` à l'état `RUNNABLE`.

Lors d'une commutation, le contexte d'exécution d'un *thread* est sauvé dans la structure *thread*. Cette structure contient aussi la pile d'exécution, des pointeurs de listes ainsi que les éléments requis par la norme POSIX.

Les sémaphores POSIX sont implémentés de façon assez similaire aux *mutex*, la principale différence réside dans la fait qu'un *thread* peut imbriquer plusieurs prise de sémaphores.

Une interruption est susceptible d'amener une modification des listes de l'ordonnanceur ou de provoquer une commutation de contexte, il faut donc s'assurer qu'aucune interruption n'arrive lors des manipulations de liste de l'ordonnanceur. Pour cela, les interruptions sont mas-

quées pendant toutes ces opérations avec deux contraintes : il faut qu'à chaque masquage soit associé un démasquage et que le temps qui sépare un masquage et un démasquage soit borné et le plus court possible. Les interruptions sont masquées aussi pendant la commutation de contexte, ce qui implique que masquage et démasquage soient réalisés par des *threads* différents. Dans le cas d'une commutation par interruption, la commutation de contexte est effectuée dans le gestionnaire d'interruption. Le retour de l'interruption peut donc avoir lieu sur un processeur différent de celui qui l'a reçu. Ce mécanisme est rendu possible par l'utilisation du processeur en mode système et par le fait que les masques d'interruptions sont identiques pour tous les processeurs. Lors de l'interruption, le contexte qui contient notamment le masque d'interruption, doit obligatoirement être sauvé dans la pile du *thread* en cours d'exécution.

8.5 AUTRES PROBLÈMES LIÉS À L'ARCHITECTURE

Accès mémoire

L'utilisation d'une mémoire partagée par l'ensemble des tâches à un coût. Toutes les tâches vont devoir accéder à des zones de mémoires qui vont grosso-modo correspondre à un tout petit nombre de composants mémoire. Cela n'est pas un problème lorsque l'on est autour d'un bus, puisque le moyen d'interconnexion lui-même va séquentialiser les accès et limiter intrinsèquement la bande passante.

Si par contre les interconnexions sont implantées à l'aide d'un réseau à large bande passante [GG00] et permettant la simultanéité de nombreux transferts, alors l'accès à cette unique mémoire risque grandement d'être le goulot d'étranglement du système en terme de bande passante.

La solution la plus évidente est d'avoir un allocateur mémoire permettant de spécifier le composant sur lequel on désire voir placé une donnée. Ceci pose un certain nombre de problèmes :

- comment spécifier cela aux fonctions de bibliothèque ? Ce besoin est particulièrement criant pour la création des *threads* pour lesquels un gain significatif peut être attendu si on répartie les piles sur plusieurs composants. Si les tâches ne migrent pas, on peut tout simplement allouer un composant RAM par processeur car ainsi on aura la garantie de ne jamais avoir d'accès concurrent aux variables locales. La norme POSIX permet cela grâce à l'utilisation des attributs. J'ai donc implanté une telle stratégie dans l'allocateur. Néanmoins, tout cela reste à expérimenter dans diverses configurations ;
- comment prendre en compte l'architecture lors de la compilation du programme ? La réponse à cette question reste encore à étudier complètement.

Accès atomiques

Les accès atomiques peuvent être implantés efficacement et indépendamment de l'interconnexion grâce à une mémoire spécialisée. Afin de minimiser le traffic, nous avons défini une cible qui a le comportement du *test and set* :

- la lecture retourne la valeur contenue à l'adresse accédée et force la valeur à cette adresse à 1 ;
- l'écriture force la valeur de la case à 0.

Ainsi, une case ne peut avoir que la valeur 0 ou 1, et la donnée protégée par ce sémaphore binaire ne doit être accédée que lorsque l'on a lu un 0. Une fois la donnée protégée modifiée, on accède en écriture au sémaphore pour le relâcher. Le logiciel doit respecter scrupuleusement

ce protocole. Le matériel nécessaire à une telle implantation est très simple – c'est typiquement un banc de registres 1 bit –, et il peut être utilisé aussi bien autour d'un bus que d'un réseau commuté.

8.6 PERFORMANCES

La simulation de systèmes précise au cycle permet de mesurer les performances du noyau pour diverse architectures. Les temps d'exécution de certaines opération, en nombre de cycles d'horloge du système, sont donnés en table.8.1. Ces mesures ont été effectués sur plusieurs systèmes comportant de 1 à 6 MIPS R3000 connectés par un PI-Bus, chaque processeur possédant des caches instructions et données. La fréquence de fonctionnement est identique pour l'ensemble du système. Les opérations mesurées sont :

- la commutation de contexte : la valeur mesurée ici représente le temps d'élection d'un *thread* plus la commutation de contexte ;
- la prise et la libération d'un *mutex* sans blocage. Dans ce cas le *mutex* n'était pas déjà pris par un autre *thread* ;
- la prise d'un *mutex* avec blocage. Le *mutex* était déjà pris. Le temps de commutation et la durée de blocage du *thread* n'est pas comptabilisé ;
- la libération d'un *mutex* avec blocage. Le *mutex* était pris par un *thread* et un seul autre était en attente. Il n'y a pas de valeur pour un processeur car, dans ce cas, la probabilité d'un tel événement est très faible ;
- la création et la terminaison d'un *thread* ;
- la prise et la libération d'un sémaphore POSIX. Le temps de commutation et la durée de blocage du *thread* n'est pas comptabilisé ;
- le gestionnaire d'interruption. Ces valeurs ont été mesurés sur quelques itérations seulement et sont présentées à titre indicatif. Tous les processeurs sont interrompus simultanément, ce qui implique une forte contention pour l'accès au code du gestionnaire d'interruption.

TAB. 8.1 – Mesure de performance, en nombre de cycles, de quelques fonctions du noyau.

	nombre de processeurs			
opérations	1	2	4	6
commutation	172	187	263	351
prise d'un *mutex* (sans blocage)	36	56	61	74
libération d'un *mutex* (sans blocage)	30	30	31	34
prise d'un *mutex* (avec blocage)	117	123	258	366
libération d'un *mutex* (avec blocage)	-	191	198	218
création d'un *thread* (1)	667	738	823	1085
terminaison d'un *thread* (1)	98	117	142	230
prise d'un sémaphore POSIX	36	48	74	76
libération d'un sémaphore POSIX	36	50	78	130
gestionnaire d'interruptions (2)	200	430	1100	1900

Les mesures ont été effectuées sur 1000 itérations de chacune des fonctions, sauf pour (1), les mesures ont été effectuées sur une centaine d'itération, et pour (2), les mesures ont été effectués que sur quelques itérations des fonctions, en interrompant tous les processeurs simultanément. Ces mesures ont été effectuées par Pascal Gomez dans le cadre de sa thèse [Gom00]. D'autres mesures mériteraient d'être effectuées en considérant des domaines d'horloge différents pour les processeurs, les mémoires, et même les coprocesseurs. Cela reste le sujet de futures experi-

mentations.

Les performances de la commutation, création de tâche et gestion d'interruption semblent très dépendantes du nombre de processeurs. En réalité, ce n'est pas le noyau qui supporte mal le passage à l'échelle, mais l'architecture qui est mal adaptée au multiprocesseur. En effet, au delà de 4 processeurs, le bus devient devient très chargé, et la latence d'accès à la mémoire est prépondérante devant le temps de calcul. C'est particulièrement criant pour le gestionnaire d'interruption, car pour cette expérience, tous les processeurs sont interrompus simultanément. Là encore, des expérimentations restent à faire pour mesurer les performances avec d'autres types d'interconnexions.

Le noyau est relativement léger, puisqu'il faut entre 5 Ko et 25 Ko selon les primitives utilisées pour son code. Chaque *thread* lui même est représenté par une centaine de mots, sans compter la pile d'exécution dont la taille peut aller de quelques mots à plusieurs kilo-mots.

8.7 CONCLUSION ET PERSPECTIVES

Le développement de ce noyau POSIX multiprocesseur, multi-tâches, multi-bancs mémoire, préemptif, symétrique, nous a permis de bien comprendre toutes les relations entre le système logiciel et la plateforme matérielle servant de support à l'exécution. Ce noyau est raisonnablement compact et performant – pour un même niveau de service et des buts similaires, il surpasse [BMR94] –. Il est une brique de base dont nous avons besoin pour expérimenter plus en avant les problèmes de conception matérielle/logicielle.

L'extension majeure que nous envisageons concerne la prise en compte de contraintes de temps, pour en faire un système temps réel. Cela nécessite la définition de nouvelles interfaces pour spécifier la période, la date butoir et le pire temps d'exécution d'une tâche. Une première étude à été réalisée par un étudiant de DEA [Ald01]. Comme on vient de le voir, ce pire temps d'exécution est une donnée du problème d'ordonnancement temps réel, et hélas, les bornes fournies par les méthodes d'estimation sont généralement très large. Nous nous intéressons donc à prévoir de manière plus précise ce pire temps d'exécution (Worst Case Execution Time) dans le cadre général des multiprocesseur, afin de maximiser l'utilisation des ressources. Cette étude a fait l'objet d'un stage de DEA [Che02]. Ces extensions sont en cours d'étude par Pascal Gomez [Gom00].

8.8 BIBLIOGRAPHIE

[CF78] Lucien M. Censier and Paul Feautrier. A New Solution to Coherence Problems in Multicache Systems. IEEE Transactions on Computers, Vol. c-27, No. 12, December 1978, pp. 1112–1118.

[BMR94] T.P. Baker, Franck Mueller and Viresh Rustagi. Experience with a Prototype of the POSIX "minimal Realtime System Profile", In *Proceedings 11th IEEE Workshop on Real-Time Operating Systems and Software,* Seattle, WA, May 1994, pp. 12–16.

[LW95] Daniel E. Lenoski and Wolf-Dietrich Weber. Scalable Shared-Memory Multiprocessing, Morgan-Kaufmann Publisher, Inc. 1995.

[LLG+] D. Lenoski, J. Laudon, K. Gharachorloo, W-D Weber, A. Gupta, J. Hennessy, M. Horowitz and M. Lam. The Stanford DASH Multiprocessor, IEEE Computer, Vol. 25, No 3, March 1992, pp. 63-79.

[Mes01] MEDEA project A502. *Multiprocessor Embedded Kernel Architectures.* `www.medea.org`, 2001.

[LvM+98] Jeroen Leijten, Jef van Meerbergen, Adwin Timmer, Jochen Jess. Stream Communication between Real-Time Tasks in a High-Performance Multiprocessor. In *Proceedings of the Design Automation and Test in Europe Conference*. Paris, France, mars 1998, pp. 125-131.

[GG00] Pierre Guerrier and Alain Greiner Architecture for On-chip Packet-switched Interconnections. In *Proceedings of the Design Automation and Test in Europe Conference*, Paris, France, Mars 2000, pp. 250-256.

[BMA02] Boris Boutillier, Laurent Mortiez et Alain Greiner. Support matériel à l'exécution multi-threads pour le processeur Mips R3000 : Micro-Architecture et caractérisation. In *Troisième Colloque du GDR CAO de circuits et systèmes intégrés*, Paris, France, Mai 2002, pp. 115-118.

8.9 PUBLICATIONS RELATIVES AU CHAPITRE 8

Conférences internationales avec comité de programme

[PG03] Frédéric Pétrot and Pascal Gomez. Lightweight Implementation of the POSIX Threads API for an On-Chip MIPS Multiprocessor with VCI Interconnect, In *Design Automation and Test in Europe, Embedded Software Forum*, Munich, Germany, March 2003. IEEE. (à paraître).

Conférences nationales avec comité de programme

[GPH02] Pascal Gomez, Frédéric Pétrot et Denis Hommais. Mutek : Un noyau Multi-tâches/Multi-processeur SMP pour systemès embarqués, *Actes du troisième colloque CAO de circuits et systèmes*, pages 107–110, Paris, France, Mai 2002.

8.10 ENCADREMENT DE THÈSE

[Gom00] Pascal Gomez *Conception et réalisation d'un noyau mutli-tâches multiprocesseur temps réel pour les systèmes intégrés matériel–logiciel.* Thèse en cours.

8.11 ENCADREMENT DE STAGES DE DEA

[Sei99] Hervé Seignol. *Implantation des threads POSIX sur machine MIPS,* Mémoire de DEA ASIME, 1999. Avec Denis Hommais.

[Ald01] Laurent Aldebert. *Prise en compte de contraintes temps réel dans un ordonnanceur*, Mémoire de DEA ASIME, 2001. Avec Pascal Gomez.

[Che02] Cyril Chevrot. *Estimation du pire cas d'exécution d'un programme séquentiel,* Mémoire de DEA ASIME, 2002. Avec Pascal Gomez.

8.12 PROTOTYPES

J'ai développé le noyau POSIX multi-tâches et multiprocesseur *mutek* (Mutli-Thread Embedded Kernel). Ce noyau est disponible sous licence GPL avec les autres outils de la plateforme Disydent. Pascal Gomez est en train de modifier profondément ce noyau pour lui ajouter des caractéristiques temps réel [Gom00].

Chapitre 9

Prospective

Les systèmes micro-électroniques embarqués ne peuvent que continuer à se développer dans les années à venir, même si par extraordinaire les limites physiques de la technologie devaient être atteintes. En effet, à partir des technologies multi-niveau de métal avec une taille de grille de transistor de l'ordre de 0.08μm la question n'est plus « comment mettre tout cela sur une puce ? » mais « comment allons nous faire pour remplir tout cet espace disponible ? ». Cette problématique s'apparente à l'évolution du logiciel, qui dans quasiment toutes les branches favorise les conceptions réutilisables avec des langages et des systèmes gourmants en ressources, par rapport à la compacité et aux performances.

Les grands projets des industriels concepteurs de systèmes nécessitent aujourd'hui des équipes de l'ordre de 100 hommes/an. Cette dimension rend très difficile la gestion des projets, et surtout ne permet plus la remise en cause de l'existant. D'autant plus que ces projets se font toujours dans l'urgence avec des pressions énormes du marketing.

La seule solution passe par la réutilisation de briques de plus en plus complexes, qui ont fait leur preuves et qui savent communiquer simplement les unes avec les autres. La communication entre briques repose à première vue sur l'acceptation par les fournisseurs et les utilisateurs d'un protocole bas niveau standard. Néanmoins, cette condition est nécessaire mais non suffisante, car la sémantique des échanges doit être connue à plus haut niveau. L'adoption au debut 2000 du standard VCI par les grandes firmes du secteur résoud en partie le problème de la communication. Cependant VCI ne permet pas de dire à partir de quel moment le contenu d'une adresse peut être lue, ou réécrite, ni par qui. L'utilisation de formalismes spécifiques à un domaine d'application est une voie, qui sans être affreusement originale, a au moins le mérite du pragmatisme.

L'évaluation qualitatives d'une architecture implantant un comportement est encore fort coûteuse en temps. Un certain nombre d'initiatives visant à permettre des simulations « fiables » au dessus du niveau cycle ont été commencées. Cependant, on se rend compte que les performances sont intimements liées au contrôle de flux, qui lui même doit être présent dans ces modèles haut niveau, ce qui n'est pas aisé. Bref, l'état des recherches n'est pas actuellement satisfaisant, et le niveau même de modélisation n'est pas clair. Nous commençons une étude sur la modélisation des architectures d'interconnexions avec Pierre Paulin et son équipe (ST Microelectronics à Ottawa). Cette étude est l'objet de la thèse d'Etienne Faure [Fau02] qui vient de commencer.

Une autre facette de la conception des systèmes sur puce est la corrélation entre l'architecture, le noyau du système, et le ou les compilateurs pour permettre l'implantation optimisée, selon des critères variés, de l'application. Les liens unissant ces trois aspects sont beaucoup

plus forts que dans les systèmes classique ou l'on a tout fait pour les rendre orthogonaux. Ces liens ont commencé d'être abordé dans une action spécifique du réseau thématique SoC du CNRS à laquelle je participe activement.

Encadrement de thèses

[Fau02] Etienne Faure. *Modélisation de performance à haut niveau des systèmes intégrés*. Thèse en cours, en co-direction avec Daniela Genius.

Chapitre 10

Conclusion

Dans l'avant propos de ce document, j'ai exprimé le fait que j'ai été amené à travailler sur de nombreux sujets ayant trait à différents aspects du domaine de la conception de circuits et de systèmes intégrés. J'espère avoir convaincu le lecteur de l'intérêt qu'il y a dans le domaine du SoC à appréhender des domaines aussi différents que le micro-noyau et la micro-architecture des circuits d'interface.

Par ailleurs, je veux croire que ce manuscript illustre clairement que le travail présenté ici n'est pas le travail d'un seul homme mais bien celui d'une équipe. J'ai commencé modestement avec Denis Hommais le développement de CASS en 1996, mais au plus fort de notre activité, liée il est vrai à la production de prototypes opérationnels pour le projet européen COSY en l'an 2000, l'équipe étaient constituée de 13 personnes : deux maîtres de conférences, quatre post-docs, deux thésards, trois étudiants de DEA et deux élèves ingénieurs. Ces prototypes ont été repris et améliorés par de nouveaux doctorants, ce qui nous permet de distribuer aujourd'hui la chaîne d'outils pour la conception de systèmes intégrés Disydent.

Aujourd'hui, l'équipe dont j'ai la responsabilité est composé d'une demi-douzaine de chercheurs (hors stagiaires). Mais nos travaux attirent autour d'eux une grande partie des chercheurs du département ASIM, comme l'illustrent les exemples ci-dessous :

- CASS ainsi que les modèles de bibliothèque existant sont utilisés pour toutes les études d'architecture, du coprocesseur flottant au processeur multi-contextes matériel en passant par le réseau commuté enfoui ;
- DPN et UGH sont utilisés par les architectes de la machine parallèle MPC, qui est un *cluster* de PCs reliés par des liens série, pour modéliser leurs composants à haut niveau puis les réaliser sur FPGA ;
- L'activité test, tirant partie de l'existance d'un processeur sur le circuit, envisage son utilisation comme testeur intégré. Hors le séquencement des opérations de test n'est pas trivial, et il est possible qu'un micro-noyau soit nécessaire pour permettre l'optimisation du temps de test.

En conclusion, et malgré leurs étendues, mes travaux forment un ensemble cohérent que je pense utile au concepteur de systèmes intégrés. Comme le souligne le chapitre 9, les axes de recherche portent d'une part sur la modélisation à plus haut niveau de systèmes matériel et logiciel et sur une meilleure intégration de l'architecture, du noyau et de la compilation. C'est à ces tâches que je compte m'attaquer à présent.

Annexes

Sont regroupés ici les informations fournies chapitre par chapitre, pour donner un résumé synthétique du travail d'encadrement, de direction de projets et d'animation de la recherche.

ENCADREMENT DE THÈSES

[Jac99] Ludovic Jacomme. *Analyse sémantique de descriptions VHDL synchrones en vue de la synthèse.* Thèse de l'UPMC soutenue en octobre 1999. En co-direction avec Alain Greiner.

[Hom01] Denis Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC soutenue en septembre 2001.

[Gom00] Pascal Gomez *Conception et réalisation d'un noyau mutli-tâches multi-processeurs temps réel pour les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC en cours.

[Buc01] Richard Buchmann. *Simulation rapide de systèmes intégrés.* Thèse de l'UPMC en cours.

[Abr01] Ana Abril. *Estimation et optimisation de la consommation dans les systèmes intégrés haute performance.* Thèse ciffre avec *Philips Digital System Labs* encadrée par Jean Gobert (PDSL), Thomas Dombek(PDSL), Habib Mehrez(LIP6) et moi-même.

[Fau02] Etienne Faure. *Modélisation de performance à haut niveau pour les systèmes intégrés.* Thèse en cours, avec iplication de *ST Microelectronics* (équipe de Pierre Paulin à Ottawa), en co-direction avec Daniela Genius.

[Don00] François Donnet. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel.* Thèse de l'UPMC en cours. Participation à l'encadrement de cette thèse dont la direction est assurée par Ivan Augé.

ENCADREMENT DE STAGES DE DEA

[Big97] Jean-François Bignolle. *Architrecture de modules de communications génériques pour PI-Bus.* Mémoire de DEA ASIME, 1997. Avec Denis Hommais.

[Pas98] Laurent Pasquier. *Modélisation cycle de modules de communications.* Mémoire de DEA ASIME, 1998. Avec Denis Hommais.

[Gom99] Pascal Gomez. *MJEPG : Conception d'un système intégré, de la spécification fonctionnelle au modèle de performance cycle true.* Mémoire de DEA ASIME, 1999. Avec Denis Hommais.

[Sei99] Hervé Seignol. *Implantation des threads POSIX sur machine MIPS.* Mémoire de DEA ASIME, 1999. Avec Denis Hommais.

[Lev99] Stéphane Levassort. *Ordonnancement à gros grain dans UGH.* Mémoire de DEA ASIME, 1999. Avec Pierre Guerrier.

[Ald01] Laurent Aldebert. *Prise en compte de contraintes temps réel dans un ordonnanceur.* Mémoire de DEA ASIME, 2001. Avec Pascal Gomez.

[Buc01] Richard Buchmann. *Interfaçage VCC/CASS/DPN.* Mémoire de DEA ASIME, 2001.

[Pet01] Boriana Petrova. *Modélisation d'un décodeur MJEPG sous VCC.* Mémoire de DEA ASIME, 2001.

[Che02] Cyril Chevrot. *Estimation du pire cas d'exécution d'un programme séquentiel.* Mémoire de DEA ASIME, 2002. Avec Pascal Gomez.

[Dup02] Yoan Dupret. *Modélisation de la consommation dans des descriptions précises au cycle.* Mémoire de DEA ASIME, 2002. En co-encadrement avec Ana Abril et Habib Mehrez.

[Fal02] Steve Falempin. *Extraction et comparaison de chemins d'exécution.* Mémoire de DEA ASIME, 2002. Avec François Donnet.

[Fau02] Etienne Faure. *Exploration et synthèse architecturale d'un décodeur MJPEG.* Mémoire de DEA ASIME, 2002. Avec François Donnet.

PARTICIPATION À DES PROJETS EUROPÉENS

COSY EP25443 : 1997-2000

Nous avions dans le projet COSY (Cosimulation and Syntesis) comme partenaire priviligié Philips Research et les Cadence European Labs. Notre contribution fut la partie matérielle de la synthèse des communication et l'outil de synthèse d'architecture guidé par l'utilisateur. Ce projet à permis de financer le développement de DPN, UGH et tout l'aspect synthèse des communications, des pilotes logiciels jusqu'au VHDL.

L'équipe travaillant sur ce projet était constitué des membres permanents suivanté : Ivan Augé (MC CNAM), Denis Hommais (Doc LIP6), Rajesh Bawa (Postdoc LIP6), Alain Houelle (Postdoc LIP6), Nicolas Vaucher (Postdoc LIP6), Ludovic Jacomme (Postdoc LIP6) et moi. Sur les trois ans du projet, l'équipe a été renforcée par en moyenne trois stages de DEA (ASIM) et deux stages d'élèves ingénieurs (CNAM) par an. J'ai eu la responsabilité scentifique et technique de cette équipe concernant nos engagements contractuels.

Médea AT-403 : 1998-2001

Dans ce projet, nos engagements concernaient la simulation cycle rapide et la traduction de sous ensemble du VHDL vers des modèles C simulables cycle par cycle. Ce projet à permis de financer les développements autour de CASS.

Hormis moi-même, ont été impliqués dans ce projet Denis Hommais (Doc LIP6) et Ludovic Jacomme (Doc LIP6).

PARTICIPATION AUX ACTIONS STIC DU CNRS

J'ai participé activement pendant un an aux réunions du Réseau Thématique SoC. J'ai œuvré dans deux actions spécifiques qui se sont achevées fin 2002 :

Bibliothèque d'IPs
: cette action est dirigée par Alain Greiner, et vise à définir une interface au niveau bit et un langage de simulation pour permettre l'intégration du comportement d'IPs développées dans les équipes françaises. Les choix arrétés sont VCI pour l'interface et un sous-ensemble de System C pour la modélisation ;

Systèmes d'exploitation pour multiprocesseurs embarqués
cette action, menée par Michel Auguin et plus prospective, vise à définir les nouvelles voies de recherches dans ce domaine. L'adéquation architecture/système/compilation et l'impact sur la consommation ont été mis en avant.

Je fais également partie du comité de pilotage du RTP Architecture et compilation nouvellement crée.

DISTRIBUTION DE LOGICIELS LIBRES

L'ensemble des développements que j'ai réalisé, que j'ai encadré et plus généralement auquel j'ai participé, est distribué sous licence GNU.

Alliance
Les structures de données et les outils décrits au chapitre 2 sont disponibles sur `www-asim.lip6.fr/alliance` ;

Disydent
Les outils constitutifs de notre environement de développement de système, décrits aux chapitres 4, 5, 6, 7 et 8 sont disponibles sur `www-asim.lip6.fr/disydent`.

COMITÉ DE LECTURE DE JOURNAUX/CONFÉRENCES

J'effectue occasionnellement des revues de papiers pour les journaux/conférences suivant :
- *Journal of System Architecture*, Elsevier ;
- *Design Automation and Test in Europe*, IEEE ;
- *Workshop on Rapid System Prototyping*, IEEE ;
- *International Conference on Microelectronics*, IEEE ;