Conception des systèmes logiciel/matériel : du partitionnement logiciel/matériel au prototypage sur plateformes reconfigurables

Frédéric Rousseau

THÈSE D'HABILITATION À DIRIGER DES RECHERCHES
Université Joseph Fourier

Spécialité : Informatique et mathématiques appliquées

Présentée par Frédéric ROUSSEAU

pour l'obtention du diplôme
d'habilitation à diriger des recherches
sur le thème :

Conception des systèmes logiciel/matériel : du partitionnement logiciel/matériel au prototypage sur plateformes reconfigurables

soutenue le 8 juillet 2005 devant le jury composé de :

Mme Dominique Borrione, professeur à l'Université Joseph Fourier, Grenoble

Mrs Flavio Rech Wagner, professeur à l'université de Rio Grande, Brésil, (rapporteur)

El Mostapha Aboulhamid, professeur à l'université de Montréal, Canada, (rapporteur)

Michel Auguin, directeur de recherche CNRS, laboratoire I3S, (rapporteur)

Eric Martin, professeur à l'université de Bretagne Sud

Ahmed Amine Jerraya, directeur de recherche CNRS, laboratoire TIMA

Remerciements

Je tiens tout particulièrement à exprimer ma profonde gratitude à Mr Ahmed Jerraya, responsable de l'équipe SLS du laboratoire TIMA, pour la confiance qu'il m'a témoignée depuis bientôt six ans. Ces conseils, son soutien, ces encouragements permanents tout au long de ces années et son dynamisme, ont joué un rôle déterminant dans ce travail. Je lui suis très reconnaissant d'avoir accepté de superviser mon HDR.

Un grand merci aux professeurs Flavio Wagner et El Mostapha Aboulhamid, qui ont répondu favorablement à ma requête pour être les "deux rapporteurs étrangers et francophones" de cette HDR. J'ai sollicité leur concours connaissant la qualité scientifique de leurs travaux et les éclairages multiples qu'ils sont susceptibles de donner sur mes propres recherches.

Des remerciements particuliers à Mr Michel Auguin qui a (encore !) accepté de rapporter sur mon travail de recherche. Déjà rapporteur de ma thèse, il suit depuis bientôt dix ans mes travaux. Je tiens à lui exprimer ma profonde gratitude pour ses précieux conseils, nos discutions diverses et variées, nos collaborations passées et futures.

Tous mes remerciements à Mme Dominique Borrione pour avoir accepté de présider ce jury. Je fus son élève à 3i, et elle est à l'origine de quelques unes de mes orientations professionnelles. Ses encouragements et ses conseils aussi bien pour l'enseignement que pour la recherche sont très précieux. Je veux lui exprimer ici ma profonde reconnaissance.

Je remercie très chaleureusement Mr Eric Martin, qui a accepté de délaisser temporairement sa charge de président de l'université de Bretagne Sud pour participer à ce jury. C'est un grand honneur qu'il me fait.

Le travail présenté dans ce mémoire est un travail collectif. Mes remerciements vont à ceux qui m'ont accompagné dans cette aventure intellectuelle depuis plusieurs années : Samy Meftali, Ferid Gharsalli, Arif Sasongko, Arnaud Grasset, Benaoumeur Senouci et Abdelmajid Bouajila. J'ai toujours apprécié leur enthousiasme et leur capacité à relever les défis que je leur lançais. J'espère leur avoir apporté au moins autant qu'ils m'ont apporté. Sans eux, ce document n'existerait pas.

Je remercie mes collègues du TIMA, les collègues de l'UJF, et plus particulièrement ceux de Polytech Grenoble, qu'ils soient personnels techniques et administratifs, chercheurs ou enseignants-chercheurs, avec qui je partage les problèmes et les plaisirs quotidiens.

Un remerciement particulier à mon collègue de bureau, Paul, qui me transmet en douceur son savoir et son expérience de l'enseignement, de la recherche et plus généralement de la vie.

Enfin, mes remerciements vont bien sur à ma famille et à mes amis qui sont toujours présents.

Prologue

Ce document résume mes activités de recherche et d'enseignement auxquelles je me suis consacré depuis une dizaine d'années, d'abord au Centre Nationale d'Etudes des Télécommunications (ex CNET devenu France Télécom R&D), puis à l'Ecole Supérieure d'Ingénieurs de Marseille (ESIM), et enfin à l'Université Joseph Fourier de Grenoble et plus particulièrement dans l'équipe SLS (System Level Synthesis) du laboratoire TIMA.

Mes travaux de recherche ont commencé avec mes travaux de thèse au CNET Grenoble en collaboration avec l'Université d'Evry Val d'Essonne sur le découpage logiciel/matériel d'applications de télécommunications. Ma recherche à l'ESIM constituait la suite de ces travaux de thèse. Depuis mon affectation comme maître de conférences au laboratoire TIMA, en octobre 1999, je m'intéresse à plusieurs aspects de la conception des parties logiciel/matériel des systèmes multiprocesseurs monopuces. Ces systèmes ont en effet introduit de nouvelles difficultés dans le processus de conception et de validation.

En parallèle, j'ai également exercé une activité d'enseignement. Depuis 1992, j'ai enseigné l'informatique, l'informatique industrielle, l'électronique, l'automatique et la conception de systèmes dans différentes écoles d'ingénieurs (ENSPG, ESIM, Polytech'Grenoble) et en Master Recherche micro-nano électronique. (ex DEA de microélectronique).

Ce mémoire présente dans sa première partie un résumé de mes travaux de recherche, menés en collaboration avec plusieurs doctorants que j'ai co-encadrés (Samy Meftali, Ferid Gharsalli, Arif Sasongko, Arnaud Grasset et Benaoumeur Senouci).

La deuxième partie du mémoire décrit mes activités et mes responsabilités administratives et collectives au sein de l'Université Joseph Fourier, mais aussi celles assurées à l'ESIM.

La troisième partie est un recueil des principales publications scientifiques de ces dernières années et elle est présentée dans un document annexe.

Résumé

Ce document retrace mes activités de recherche depuis ma thèse soutenue en juillet 1997. Certains des travaux présentés sont achevés, d'autres sont en cours ou encore dans un stade exploratoire.

De 1993 à 1999, je me suis intéressé aux différents aspects du partitionnement logiciel/matériel dans la conception de systèmes intégrés numériques de télécommunications. Depuis 1999, mes travaux ont porté sur la conception de systèmes multiprocesseurs monopuces, et plus particulièrement sur ce qui a trait aux relations entre logiciel et matériel. Ces systèmes sont généralement dédiés à une application ou à une classe d'applications, ce qui permet d'optimiser l'architecture et les programmes. Mes recherches ses sont donc focalisées sur l'architecture mémoire, les interfaces de communication entre composants et le prototypage. Pour ces trois axes de recherche, des méthodes et des outils d'aide à la conception ont été définis et développés.

Des travaux toujours en cours portent sur la généralisation d'une méthode de conception de composants d'interface matériels à partir d'une spécification sous forme de services requis et fournis. Une telle spécification est déjà utilisée pour représenter des protocoles dans les réseaux de communication et pour le développement des couches logicielles de communication. Son extension à la conception des interfaces matérielles homogénéiserait les langages, méthodes et outils de l'environnement de conception.

Mes travaux futurs s'orientent vers deux axes : L'intégration logiciel/matériel et l'adéquation entre architecture et système d'exploitation. Dans les deux cas, les relations étroites entre les ressources physiques de l'architecture et les couches logicielles qui y accèdent doivent permettre d'améliorer sensiblement les performances.

Abstract

This document describes my research activities since my PhD defended in July 1997. Some works presented here are complete, some others are still ongoing or at a preliminary stage.

From 1993 to 1999, I worked on different aspects of hardware/software partitioning for digital integrated system design. Since 1999, my research topics have been focussed on MultiProcessor System on Chip (MPSoC) design, and more precisely on links and relations between hardware and software. As these systems are specifically designed for an application, this allows both the architecture and the software to be optimized. Research topics concern memory architecture, communication interface design and prototyping, and for all of them, aided methods and tools have been defined and developed.

Some ongoing work deals with the generalization of a hardware interface design method, using required and provided services as specification. Such a method is already used to describe protocols in communications networks and to develop communications software. Our objective is the unification of languages, methods and tools, in a design environment.

Hardware/software integration and appropriateness of architecture and operating system will be my next research topics. In both cases, close relations between the physical resources of the architecture and the software layers could allow significant performance improvements.

Table des matières

Table des figures

I La conception des systèmes logiciel/matériel

I.1 Contexte, motivation

Le terme "Codesign" est apparu au début des années 1990 pour marquer une nouvelle façon de penser la conception des circuits intégrés et systèmes. La "conception conjointe du logiciel et du matériel" était devenue nécessaire pour répondre aux exigences du marché des systèmes intégrés. En effet, l'émergence des systèmes multimédia (téléphones portables, consoles de jeu, …) entraînait une plus grande complexité de la partie électronique et la concurrence économique imposait un temps de conception encore plus court.

Une solution pour diminuer le temps de conception des systèmes était d'utiliser des machines programmables à base de microprocesseurs, puisque la conception de matériels spécifiques (circuits intégrés ASIC) n'était plus à faire. Cette conception (modélisation du circuit, plan de masse, routage, gravure de la plaquette sur silicium, test, ...) se révélait en effet plus longue que la production de logiciel (code de programmation) et également beaucoup moins souple en cas de modification.

Aussi, pour concevoir un système d'un coût raisonnable tout en respectant les performances imposées, les concepteurs s'orientaient vers une approche mixte. Une partie était réalisée avec des composants programmables, c'est la partie logicielle. L'autre partie était réalisée avec des composants matériels spécifiques à l'application, c'est la partie matérielle. L'utilisation conjointe de ressources logicielles et matérielles nécessitait de nouvelles méthodes de conception pour trouver le meilleur compromis entre parties logicielles et matérielles (partitionnement logiciel/matériel) et pour permettre leur conception simultanément.

Dans les années 1990, de nombreuses équipes de recherches, notamment aux Etats-Unis, mais aussi en France (laboratoire TIMA, CNET Grenoble et Université d'Evry, I3S, LESTER, …) se sont penchées sur le problème du partitionnement logiciel/matériel. Des techniques et algorithmes ont permis de résoudre le problème dans des cas particuliers (systèmes orientés flot de données ou flot de contrôle), mais en se limitant généralement à des architectures très simples (monoprocesseurs). Néanmoins, plusieurs outils spécifiques et liés à une plateforme de simulation (ou d'émulation) ont montré l'intérêt d'aider le concepteur dans cette étape de conception. L'automatisation permettrait de guider le concepteur pour décider du partitionnement. On doit malheureusement dire que le problème n'est pas résolu et reste identique à ce jour. D'une part, le problème est extrêmement difficile et dépend de paramètres technologiques (vitesse, consommation, …), de l'application (architecture), de paramètres économiques (coût de conception et fabrication) et de paramètres "sociologiques" (sécurité, maintenabilité, testabilité, …). Ces derniers sont difficiles à formuler mais aussi difficilement quantifiables et mesurables, contrairement aux autres paramètres. De plus, ils évoluent, ce qui remet en cause les algorithmes et solutions trouvées il y a une dizaine d'années. D'ailleurs aucun outil commercial n'existe à ce jour.

Cette façon de penser et de voir un flot de conception séparé pour logiciel et matériel a rapidement fait apparaître deux nouvelles difficultés : la nécessité de concevoir des interfaces spécialisées et optimisées pour faire communiquer logiciel et matériel, et l'intégration et la validation des composants après leur conception.

En fait ces deux difficultés sont étroitement liées, et à l'époque du Codesign, peu de laboratoires s'y intéressaient. Mais l'idée était de traiter d'abord le problème du partitionnement qui

représentait une étape de plus haut niveau et de voir ensuite comment résoudre les problèmes de bas niveau, c'est-à-dire l'intégration des parties logicielles et matérielles.

La difficulté liée à la conception des interfaces est devenue encore plus préoccupante à la fin des années 1990 avec l'apparition des "systèmes multiprocesseurs monopuces" intégrant sur une même puce de silicium des composants de nature différente (processeur, mémoire, composants spécifiques IP ("Intellectual Properties"), interconnexions, …) (Figure 1). La conception de tels systèmes repose sur l'assemblage de composants existants, ce qui suppose que ces composants aient des interfaces physiques et des protocoles de communication compatibles. Sinon, une adaptation est nécessaire, et elle requiert alors un composant matériel d'interface de communication.

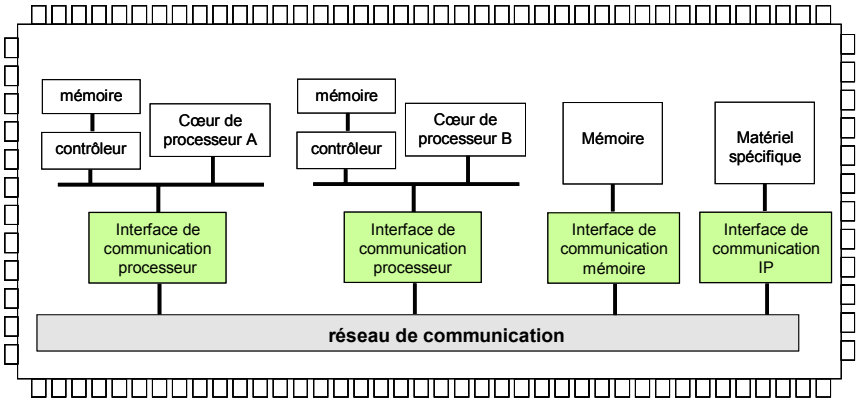

Figure 1 : Description d'un système monopuce

Mais les applications ont suivi l'évolution de la technologie permettant cette intégration, et pour les besoins des applications multimédias aux fonctions multiples, le besoin en logiciel s'est encore accru. Pour certaines applications, il représente plusieurs centaines de milliers de lignes de programme, ce qui correspond à 70% du temps de conception d'un système. On a alors introduit des systèmes d'exploitation pour gérer la synchronisation des tâches de ces programmes complexes et les ressources physiques. La structure en couches de la partie logicielle est apparue pour les besoins de la conception, là aussi, en séparant la couche de programmes applicatifs de la couche bas niveau de gestion des ressources matérielles (Figure 2).

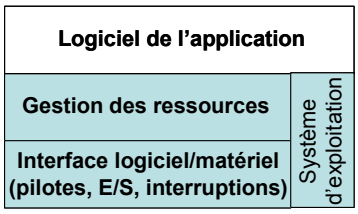

Figure 2 : Structure en couches de la partie logicielle

Cette structuration en couches permet de rendre le logiciel de l'application indépendant de l'architecture matérielle sur laquelle il sera exécuté. La partie interface et communication est alors reportée dans la couche du système d'exploitation, et plus particulièrement dans une couche interface logiciel/matériel. Il s'agit là de pilotes logiciels en interaction avec les interfaces matérielles placées entre le processeur et le réseau d'interconnexion.

La recherche dans le domaine de la conception des interfaces logiciel/matériel est abondante. Elle est un des points critiques de la conception, notamment pour la validation des communications qui préfigurent des performances du système. C'est aussi une des parties à concevoir pour chaque élément de l'architecture, ce qui est long et fastidieux. Des outils automatiques d'aide à la conception faciliteraient la tâche des concepteurs.

I.2 Evolution vers la conception de systèmes multiprocesseurs monopuces

L'architecture globale de ces systèmes est donnée sur la Figure 3. Elle est décomposée en couche afin de maîtriser la complexité pour la partie matérielle et pour la partie logicielle. La partie matérielle est décomposée en deux couches. La couche basse contient les principaux composants (processeurs, composants spécifiques, mémoires). La couche de communication matérielle embarquée est un ensemble de dispositifs nécessaires à l'interaction entre les composants (réseau de communication complexe). Il convient alors d'ajouter des couches d'adaptation entre le réseau de communication et les composants de la première couche. Les trois couches de la partie logicielle sont celles décrites précédemment. Un tel modèle induit des changements dans la méthode de conception.

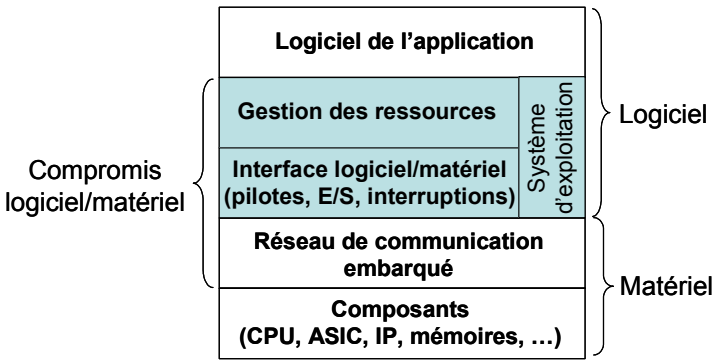

Figure 3 : Architecture des systèmes multiprocesseurs monopuces

Au niveau de l'architecture matérielle, les composants de base sont séparés de la couche de communication. Dans les schémas traditionnels, le travail de l'architecte consiste à tailler des composants sur mesure afin de prévoir les interconnexions de manière efficace et d'obtenir les meilleures performances. Ce schéma n'est plus applicable à partir d'un certain degré de complexité. Aussi pour les systèmes monopuces, le travail de l'architecte consiste à assembler des composants existants en vue de respecter des contraintes de performances et de coûts au niveau du système global et non plus au niveau du seul composant. Ainsi la valeur ajoutée de la conception des systèmes monopuces se situe plutôt au niveau de la couche de communication.

Au niveau de l'architecture logicielle, la complexité des applications impose une décomposition du code en couches et même l'utilisation d'un système d'exploitation. Ainsi on maintient la séparation des métiers pour le développement du logiciel et du matériel.

Au niveau de l'architecture globale, les choix des composants de base et du modèle de communication ne peuvent plus être faits sans prendre en compte l'organisation du logiciel. Les compromis logiciel/matériel fixent les grands choix d'architecture ainsi que les limites entre les couches de communications matérielles et les couches basses de communication logicielles. En effet, l'existence de MMU, DMA et autres facilités matérielles pour les entrées/sorties simplifie la couche de communication logicielle.

Enfin, dans le processus de conception, ces systèmes nécessitent la coopération de quatre types de métier :

- le concepteur système qui définit l'architecture globale, réalise les choix de partitionnement logiciel/matériel et les architectures du réseau de communication, choisit les algorithmes et fixe les performances,

- le concepteur de logiciel écrit les différentes couches logicielles,

- le concepteur de matériel conçoit la ou les parties matérielles,

- l'intégrateur est chargé de faciliter la communication enter le concepteur du logiciel et celui du matériel. C'est généralement lui qui réalise la couche basse du logiciel.

Le fait que les couches de communication soient réalisées par des équipes différentes peut entraîner des surcoûts dus aux précautions et/ou aux sous-utilisations des ressources. Aussi, ce métier doit évoluer et comporter la conception de toutes les couches de communications. Cette évolution permet la séparation totale entre la conception des couches de communications et les parties matérielles et logicielles. Une telle séparation est nécessaire pour introduire un peu de flexibilité et de modularité dans l'architecture en changeant plus facilement de modules de base. Le fait de décrire l'architecture globale du système et d'abstraire la communication entre les composants doit permettre de mieux finaliser les interfaces entre les différentes couches à l'aide d'interfaces de haut niveau.

On comprend aisément aussi la difficulté à créer des outils automatiques de conception, car il faut des outils d'aide pour chacun des types de métier.

I.3 Contributions

Mes recherches autour des systèmes logiciel/matériel visent à définir des méthodes et à développer des techniques et des outils d'aide à la conception et à la validation. Pour cela, je me suis intéressé aux techniques de découpage logiciel/matériel, puis à différents aspects d'architecture des systèmes liés aux interactions logiciel/matériel, tels que l'architecture mémoire et les interfaces de communication. Enfin, le prototypage sur une plateforme reconfigurable est une technique efficace de validation de ces systèmes, mais le passage de la spécification au prototype est un processus long et difficile. Mes travaux ont pour objet de faciliter l'obtention de ce prototype.

I.4 Plan du mémoire pour la partie recherche

La suite de ce mémoire comporte six chapitres. Le chapitre II introduit un flot classique de conception et de vérification de systèmes, et détaille le flot ROSES et les outils associés développés dans l'équipe SLS du TIMA. Le chapitre III rappelle les principes de base du partitionnement logiciel/matériel, une des étapes clés dans la conception des systèmes, et donne quelques directions sur les travaux liés aux algorithmes et méthodes de partitionnement. Le chapitre IV résume les travaux sur la recherche d'une architecture mémoire efficace pour les systèmes multiprocesseurs monopuces, et décrit une méthode d'exploration développée pour ROSES. Pour connecter ces composants mémoire au reste du système, il est nécessaire d'adapter leur interface physique et les protocoles d'accès au reste du système, ce qui requiert un composant d'interface matériel. Le chapitre V explique l'architecture générique de ces composants d'interface, leur génération automatique et aborde la génération des programmes pour les piloter. Le chapitre VI concerne le prototypage et présente notre vision d'un prototypage sur une plateforme reconfigurable. Enfin, le chapitre VII conclut ces travaux et donne quelques perspectives.

II Flot de conception et de vérification des SoC

II.1 Introduction

Pendant de nombreuses années, chercheurs et industriels se sont focalisés sur les problèmes de conception et de représentation aux niveaux physique puis logique des éléments de base des circuits intégrés (transistors, portes logiques) qui constituent les plus bas niveaux d'abstraction. Les outils informatiques d'aide à la conception se sont développés conjointement (simulateurs logiques, outils de synthèse) et font maintenant partie intégrante du processus de conception de circuits.

Les applications étant de plus en plus complexes, et le temps de mise sur le marché ("time to market") devant être toujours plus court, chercheurs et industriels se focalisent maintenant sur des niveaux d'abstraction plus élevés, dans lesquels les concepteurs manipulent un nombre réduit d'objets. La conception est ainsi plus rapide, et la partie bas niveau est réalisée par des outils automatiques.

La conception de telles applications à un niveau d'abstraction élevé permet de s'affranchir des détails de réalisation, en ne conservant que des informations sur le codage du comportement. A un tel niveau, le concepteur peut traiter des applications plus complexes, et choisir la technologie qui réalise le mieux chacune des parties du système tout en respectant les contraintes sans se préoccuper de la description explicite du niveau physique des ressources utilisées.

II.2 Flot de conception général

La conception des systèmes multiprocesseurs monopuces se décompose en plusieurs étapes (Figure 4) et commence avec une description à un haut niveau d'abstraction [ZER02] pour s'affranchir des nombreux détails de réalisation.

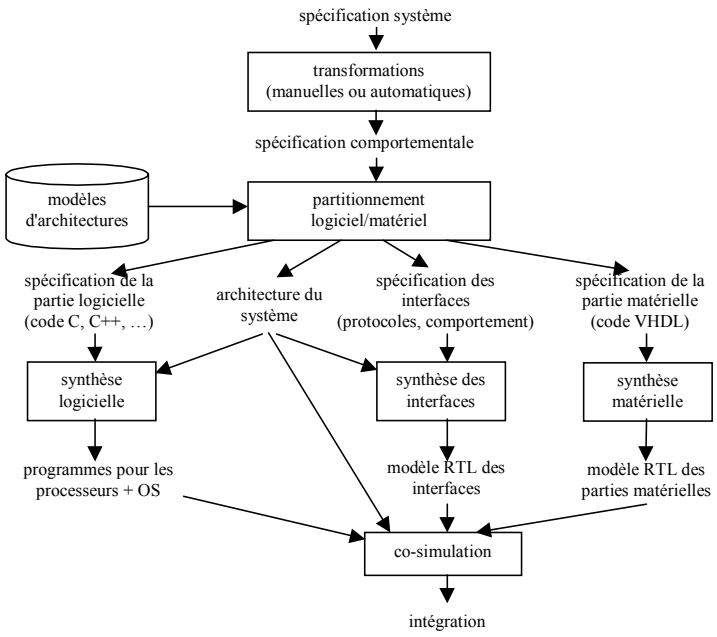

Figure 4 : Flot idéal de conception logiciel/matériel

A ce niveau d'abstraction, on s'intéresse à la fonctionnalité, indépendamment de l'implémentation finale, ce qui correspond à la conception système. On recherche des algorithmes et des représentations de données les plus adéquates. On obtient une spécification fonctionnelle que l'on valide généralement par simulation.

L'étape suivante consiste à trouver une architecture efficace. C'est l'exploration d'architectures qui détermine la réalisation logicielle ou matérielle de tous les composants. Grossièrement, les composants qui nécessitent des performances élevées sont réalisés en matériel, et ceux qui nécessitent de la flexibilité sont réalisés en logiciel. On choisit aussi dans cette étape les composants physiques (processeur, DSP, …), qui exécutent les parties logicielles, ainsi que l'architecture mémoire, la gestion des E/S, …. A la fin de cette étape, on obtient les spécifications de tous les composants matériels et logiciels.

Les étapes suivantes sont la conception du matériel, du logiciel et des interfaces composées elles aussi de parties matérielles et logicielles. Le principe est de réutiliser des composants existants, ce qui permet de gagner du temps par rapport à une conception complète de tout le système. Pour les composants matériels qui n'existent pas déjà, la conception peut suivre le flot traditionnel de conception avec les différentes étapes de synthèse (comportementale, logique puis physique). Le logiciel est implémenté en couche pour séparer les différentes fonctionnalités. Au plus bas niveau, des pilotes ou interfaces logicielles ("Hardware Abstraction Layer") permettent d'accéder aux ressources matérielles. Au dessus, le système d'exploitation gère l'exécution des différentes tâches de l'application, ainsi que les E/S. Enfin le code de l'application s'exécute à travers ce système d'exploitation (Figure 2). Une phase traditionnelle de compilation, édition de liens et chargement permet d'obtenir le code exécutable sur les processeurs cibles.

La dernière phase est la phase d'intégration logiciel/matériel. Il s'agit d'une part de vérifier que le logiciel s'exécute correctement sur les composants programmables, mais aussi que les échanges d'informations entre les composants sont corrects. Pour ces échanges d'informations, des composants d'interface (adaptateurs de communication) sont placés entre les composants et le réseau de communication pour adapter les différents protocoles et le type de données. Entre un processeur et le réseau de communication, ces adaptateurs de communication peuvent être complexes avec une partie logicielle (pilotes du processeur) et une partie matérielle (composants d'interface). La réalisation de ces composants d'interface est une des difficultés de la conception des systèmes multiprocesseurs monopuces, et leur génération automatique est un des axes de recherche de l'équipe SLS du laboratoire TIMA. Ceci facilite l'exploration d'architectures, accélère la conception, et réduit les erreurs lors de la conception de ces interfaces.

II.3 Flot de conception en utilisant ROSES

Le flot ROSES du groupe SLS permet la génération automatique d'interfaces logiciel/matériel des systèmes monopuces. L'interface logiciel/matériel (Figure 5) générée est un adaptateur de communication entre les tâches exécutées par le processeur et les ressources matérielles. Plus précisément l'interface logicielle permet au code de l'application d'accéder aux ressources matérielles du processeur (typiquement pour faire des E/S) et l'interface matérielle permet au processeur d'accéder au réseau de communication.

Les interfaces matérielles prises en compte par ROSES concernent les processeurs [LYO03], mais aussi la mémoire [GHA03], et les composants matériels spécifiques [GRA04]. Les interfaces logicielles sont les systèmes d'exploitation [GAU01] et les couches de communications [PAV04]. L'intérêt de ROSES est la génération automatique des interfaces à partir d'une spécification du système et de quelques caractéristiques pour guider la conception (protocoles, adressage, et autres paramètres). De plus, ce flot permet une validation à plusieurs niveaux d'abstraction à l'aide de mécanismes de co-simulation [NIC02].

Le principe le plus important dans ce flot est la conception d'un système complet par assemblage d'éléments [CES02]. Que ce soit pour composer les parties logicielles des interfaces ou les parties matérielles mais aussi le développement des modèles de simulation, la technique reste la même : elle consiste en un assemblage d'éléments de bibliothèque.

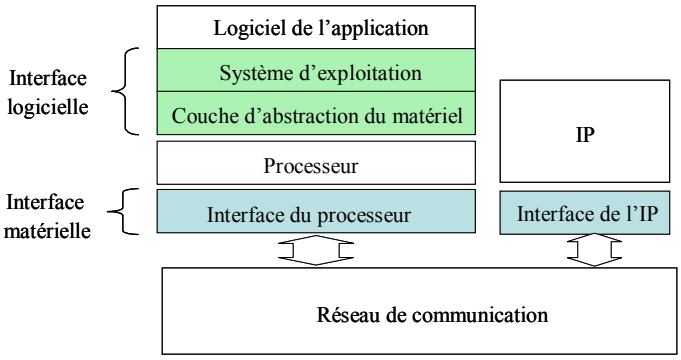

Figure 5 : Interfaces logiciel/matériel dans un système monopuce

Le flot de conception ROSES est composé principalement de trois outils : ASAG, ASOG, et Cosimix.

- ASAG ("Application Specific Architecture Generator") est l'outil de génération des interfaces matérielles,
- ASOG ("Application Specific Operating system Generator") est l'outil de génération des interfaces logicielles,
- Cosimix est l'outil de génération des interfaces pour simulation.

La Figure 6 résume le flot qui permet la génération d'interfaces à partir d'un système décrit au niveau architecture vers une description au niveau RTL. L'entrée du flot est décrite en langage de spécification développé par le groupe SLS. Ce langage est une extension de SystemC [SYS00] nommé VADeL ("Virtual Architecture Description Language").

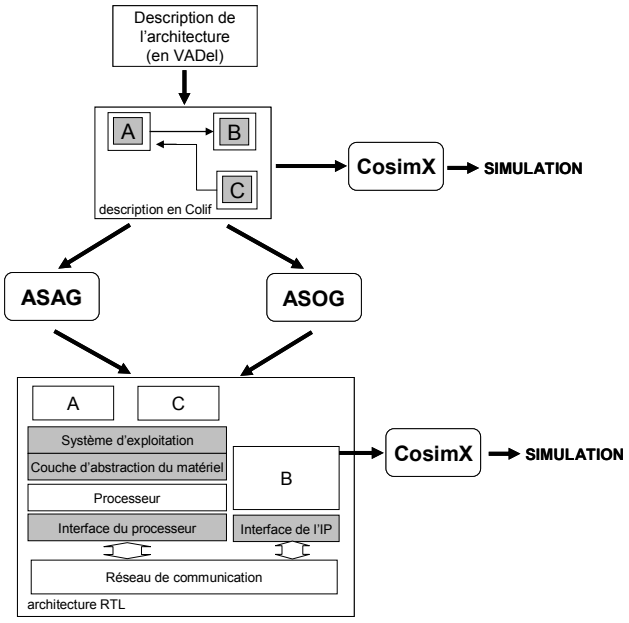

Figure 6 : Le flot ROSES

VADel [CES01] est un langage construit à partir de C++ : il est donc orienté objet. Le C++, avec son concept de classes d'objets permet de développer facilement de nouvelles structures de données. En fait, VADeL n'est pas une extension directe de C++, mais c'est une extension de SystemC, un langage de description de matériel pour la simulation.

La première étape du flot consiste à traduire la spécification en VADel annotée en COLIF [CES01]. La spécification en COLIF décrit un système comme un ensemble d'objets interconnectés de trois types : les modules, les ports et les liens (en anglais : net). Un objet, quel que soit son type, est composé de deux parties. La première partie, nommée "entity", permet la connexion avec les autres objets. La deuxième partie, nommée "content", fournit une référence à un comportement ou des instances d'autres objets. Cette possibilité d'instancier des éléments permet le développement de modules, ports et liens hiérarchiques.

A partir de ce modèle, vient une étape de génération d'interfaces matérielles (outils ASAG). Un autre outil (ASOG) permet la génération des interfaces de communication logicielles. Il existe aussi un outil de génération d'interface de simulation utilisable à partir du modèle haut niveau et de modèle raffiné (CosimX).

ASOG a été développé par Lovic Gauthier [GAU01]. Cet outil génère l'interface logicielle qui consiste en un système d'exploitation minimal et la couche d'abstraction du matériel à partir de paramètres donnés dans les ports en COLIF. Ces paramètres décrivent les services de communication demandés et requis par la tâche.

ASAG a été développé par Damien Lyonnard [LYO03]. Cet outil prend en entrée, la description structurelle de l'application en COLIF. ASAG possède deux bibliothèques. L'une contient des structures génériques d'interfaces et l'autre des fichiers décrivant le comportement des composants assemblés. En sortie, on obtient une description COLIF au niveau RTL de l'architecture de l'interface et de celle de l'architecture locale du processeur. On obtient aussi le code synthétisable VHDL ou SystemC de l'architecture.

CosimX, développé par Gabriela Nicolescu puis par Adriano Sarmento, est utilisé pour valider les systèmes décrits à plusieurs niveaux d'abstraction. Il permet de rendre exécutable la description VADeL du système. L'exécution du système avant génération des interfaces permet de valider le comportement des tâches, des IP ainsi que le choix des services de communication. CosimX offre aussi la possibilité de tester le fonctionnement du système après le raffinement partiel ou complet du système (co-simulation).

II.4 Evolution du flot de conception pour les architectures mémoires

La description VADel ne fait pas apparaître de façon explicite certains des composants qui composent le système. Typiquement, les composants de mémorisation sont implicites ; les mémoires associées aux processeurs (pour les programmes et les données) ou les mémoires pour les informations à partager (images, son, ….). Or l'architecture mémoire a une réelle importance sur les performances globales du système.

Les travaux de Samy Meftali [MEF02b] sur l'exploration d'architectures mémoire ont été intégrés dans le processus de raffinement du modèle COLIF, pour produire un nouveau modèle COLIF avec des modules mémoires explicites. L'intérêt est de faire apparaître les transferts de données entre composants pour avoir une idée plus réaliste des charges de données sur les media de communication.

II.5 Conclusion

Ce flot ROSES correspond à toutes les étapes du flot de conception de la Figure 4 à l'exception de la conception du matériel, qui doit suivre un processus standard, mais non intégré dans ROSES. La conception de niveau système est décidée manuellement, mais dans l'environnement COLIF. L'exploration d'architectures est aussi une étape de décision du concepteur (avec une aide pour l'architecture mémoire) mais les outils ASAG, ASOG et Cosimix permettent de rapidement évaluer

ou simuler le résultat. La conception des interfaces est le point clé de ces travaux, et l'environnement de simulation et de prototypage valide l'intégration logiciel/matériel.

D'autres travaux existent et proposent une méthodologie et des outils pour la conception des systèmes multiprocesseurs monopuces. On peut notamment citer les travaux de l'Université de Montréal [ABO03] [BOI04]. Leur solution est basée autour de .NET, standard de génie logiciel proposé par Microsoft et devenu un standard ISO. ESys.NET est donc un outil de modélisation au niveau système et un environnement de simulation.

II.6 Publications relatives au flot de conception des SoC

[RN2] F. Rousseau, *Conception des Systèmes VLSI*, E 2 455, ouvrage de base édité par les Techniques de l'Ingénieur, fév. 2005.

III Le partitionnement logiciel/matériel

Les travaux sur le partitionnement logiciel/matériel ont débuté avec mon travail de thèse en oct. 1993. Et je suis resté très actif dans ce domaine de recherche jusqu'en 1999 avec de nombreuses publications. Un travail de synthèse a fait l'objet d'un chapitre de livre publié chez Hermès.

III.1 Nécessité et principe de base du découpage logiciel/matériel

Le découpage ou partitionnement logiciel/matériel ("hardware/software partitioning") est une phase importante de la conception de systèmes (et plus particulièrement de la phase d'exploration d'architectures) et consiste à rechercher le meilleur compromis logiciel/matériel. C'est dans cette phase, que sont effectués les choix menant à une réalisation soit matérielle, soit logicielle des différentes parties constituant le système. En règle générale, le logiciel est utilisé pour réduire les coûts de conception et le matériel pour augmenter les performances. De nombreuses techniques et algorithmes ont été proposés pour aider le concepteur dans cette tâche. Le but ultime étant bien sûr d'automatiser cette tâche.

La spécification comportementale étant découpée en sous fonctions, le choix de leur réalisation, matériel ou logiciel, se pose. Mais comment choisir entre logiciel et matériel ? Les choix effectués dans cette étape de partitionnement sont des choix définitifs qui ne sont plus remis en cause dans les étapes suivantes de conception. Un mauvais choix à ce niveau impose alors de recommencer le cycle de conception du partitionnement jusqu'à la co-simulation. Il faut donc prêter une grande attention à cette étape.

Dans le contexte de la conception système, le logiciel peut être défini comme "une séquence d'opérations s'exécutant sur une machine programmable nécessaire au fonctionnement d'un ensemble de traitements de l'information". Les machines programmables sur lesquelles est exécuté le logiciel sont de différents types : processeur général, processeur de traitement du signal (DSP), cœur de processeur, ….

Le matériel est défini par opposition au logiciel. C'est "une structure physique non programmable". Sa fonctionnalité est figée lors de la conception. Il n'est pas possible, comme pour le logiciel, d'effectuer une modification pour obtenir une nouvelle fonctionnalité. Un ASIC est du matériel, car ce circuit spécifique est conçu pour une application donnée.

Il faut noter que certains composants sont à la frontière entre logiciel et matériel. Il s'agit des composants matériels logiques programmables tels que les FPGA ("Field Programmable Gate Array"). Les fonctions sont figées lors de la conception, mais la programmation des interconnexions modifie la fonctionnalité.

III.2 Techniques de partitionnement

III.2.1 Méthode d'optimisation

Cette étape dans la conception de systèmes consiste généralement à appliquer un algorithme d'optimisation sur l'ensemble des sous fonctions de la spécification. On cherche alors à minimiser un critère (ou un ensemble de critères) donné tel que la surface, le temps d'exécution, la consommation ou autres. Ceci suppose de maîtriser les problèmes d'estimation (ou d'évaluation) de ces critères pour une ou plusieurs architectures cibles. L'algorithme de partitionnement est donc un algorithme d'optimisation (ou de minimisation), cherchant une ou des réalisations

optimisées pour un problème donné. Une excellente introduction à de telles heuristiques est faite dans [SIA02]. On trouve aussi dans [LOP03] une comparaison entre plusieurs algorithmes de minimisation appliqués au partitionnement logiciel/matériel.

Bien sur, ce choix repose sur les caractéristiques des différents types de composants utilisés dans l'architecture cible visée. Le processeur standard offre la souplesse de programmation et le composant spécifique donne les temps de traitement les plus courts. Il existe des solutions intermédiaires basées sur des cœurs de processeurs ou des processeurs spécialisés offrant des compromis souplesse-performance satisfaisants pour une large classe d'applications.

III.2.2 Algorithmes et outils de partitionnement

La fin des années 1990 a vu de nombreux travaux sur le partitionnement logiciel/matériel, que se soit en terme de méthodes et d'outils ou d'algorithmes [JER02a][NIE98]. On peut citer les travaux les plus marquants, avec les outils VULCAN de l'université de Stanford, COSYMA de l'université de Braunschweig, les travaux de l'université d'Irvine avec les SpecCharts, la méthode basée sur le langage UNITY de l'université de Karlsruhe et de l'UFPE (Brésil). Quelques algorithmes ont aussi vu le jour : GCLP de l'université de Berkeley, BCS de l'université d'Irvine, des algorithmes de programmation en nombres entiers, des travaux autour d'un langage de spécification LOTOS, des travaux autour du compilateur GNU C pour partitionner d'une application décrite en C, des algorithmes de partitionnement basés sur des algorithmes d'ordonnancement issus de la synthèse d'architectures, en tenant compte ou non des communications entre les parties matérielles et logicielles, algorithmes génétiques, …

III.2.3 Interfaces de communications

La communication entre les différentes parties du système est facilement négligée lors de la conception système, mais elle est souvent une ressource critique dans les systèmes embarqués, que se soit en terme de surface ou de performance. Les outils automatiques de génération d'interfaces logiciel/matériel ne sont pas très nombreux [JER02a]. Les premiers travaux dans ce domaine étaient les outils COSMOS du TIMA [BAG02], POLIS de l'université de Berkeley. Avec CoWare [VER98], les processus communiquent via des ports et des protocoles. Cet outil génère alors les parties logicielles (pilotes d'E/S) et matérielles (logique d'interfaces) pour réaliser la communication. Ces interfaces tiennent compte des spécifications (performances) et des caractéristiques des composants qui réalisent les parties logicielles et matérielles.

L'équipe du TIMA poursuit ces travaux autour des interfaces de communications, logicielles et matérielles, appliqués au domaine des systèmes multiprocesseurs monopuces, qui font l'objet du chapitre V.

III.2.4 Estimateurs

Des estimateurs sont nécessaires dans l'étape de partitionnement. En effet, les valeurs données par les estimateurs renseignent le concepteur sur la qualité des solutions trouvées, afin de prédire les résultats de la conception sans aller jusqu'à la réalisation totale. C'est une estimation rapide des performances ou des caractéristiques de l'architecture.

Le problème est alors de trouver des métriques pour évaluer les solutions, et ces métriques se limitent généralement aux critères technologiques.

En matière d'estimation, trois types de valeurs sont nécessaires : des valeurs de performance (nombre de cycles d'horloge, temps d'exécution, taux de communication), des valeurs de coût (surface, coût de fabrication pour le matériel, taille du programme et de la mémoire pour le logiciel), et d'autres valeurs diverses telles que la puissance, la testabilité, le temps de conception.

Ces valeurs donnent une estimation de qualité de la solution, mais certaines sont difficiles à évaluer.

Les méthodes d'estimations que l'on trouve dans la littérature peuvent être classées dans trois catégories: statiques, dynamiques et mixtes :

- *dynamique* : la mesure de performance d'une architecture est le résultat de l'exécution d'un modèle (exemple : simulation).

- *statique* : l'estimation de performance d'une architecture est le résultat d'une analyse statique d'une spécification (exemple : analyse de chemins dans une spécification de flot de contrôle).

- *mixte dynamique/statique* : c'est l'utilisation de quelques éléments des deux approches précédentes pour l'analyse de performance d'une architecture.

Les approches dynamiques sont en général très précises. Leur inconvénient majeur est le temps nécessaire pour l'obtention du modèle à simuler (synthèse, génération, compilation), ainsi que le temps de la simulation. Ce qui les rend, en pratique, inutilisable dans le contexte particulier de l'exploration où le nombre de modèles à analyser est énorme. D'un autre coté, les approches statiques sont certes très rapides (pas de génération de modèles à simuler, ni de simulation), mais les tâches de modélisation et d'estimation sont complexes à cause de la distance qui sépare les concepts de spécification de l'implémentation.

Nous trouvons dans la littérature très peu de travaux qui visent l'analyse de performances en vue de l'exploration d'architectures pour la conception conjointe matérielle/logicielle. Cependant, beaucoup de travaux ont été réalisés pour résoudre des problèmes séparés comme l'estimation du temps d'exécution du logiciel, l'analyse de performance des circuits ASIC ou encore l'architecture de systèmes complexes.

L'estimation de performance peut être faite sur plusieurs niveaux d'abstraction. En effet, dans un environnement de conception système, nous partons d'un niveau d'abstraction système pour arriver au niveau d'implémentation RTL. Au niveau RTL, l'analyse de performance se caractérise par une grande précision, mais elle consomme beaucoup de temps. En remontant dans les niveaux d'abstraction, le temps de l'analyse de performance diminue, mais la précision diminue également. Aussi, avec l'écart important entre les deux niveaux d'abstraction, système et RTL, l'analyse de performance au niveau système peut devenir très imprécise voire inexploitable.

III.3 Conclusion

Depuis le début des années 90, le nombre de publications relatif au domaine du partitionnement logiciel/matériel a beaucoup baissé sans que ce problème soit résolu. En fait, le problème est très complexe et dépend généralement de l'application et du moment. L'apparition d'un nouveau composant peut remettre en cause un partitionnement logiciel/matériel effectué quelques jours plus tôt. Par exemple, les FPGA dynamiquement reconfigurables entraînent une nouvelles problématiques [BEN02].

De plus, les premiers travaux concernaient des architectures cibles relativement simples composées d'un processeur et de parties matérielles (ASIC), le tout étant interconnecté par un bus ou un réseau de communication simple. L'apparition ces dernières années d'applications complexes nécessite des architectures plus performantes : un seul processeur ne suffit plus. Il faut alors réaliser le système avec plusieurs processeurs, plusieurs circuits existants et un réseau de communication performant entre tous ces composants. Le partitionnement des différentes fonctionnalités sur une telle architecture devient beaucoup plus difficile, puisqu'il ne s'agit plus de choisir uniquement entre logiciel et matériel, mais sur quel logiciel et avec quel matériel.

III.4 Principales publications relatives au partitionnement logiciel/matériel

[O2] F. Rousseau, *La conception système et le partitionnement logiciel/matériel,* chapitre de l'ouvrage : Conception de haut niveau des systèmes monopuces, traité EGEM Electronique, Hermès Science Publications, mai 2002, ISBN 2-7462-0433-9.

[O3] M. Aiguier, J. Benzakki, G. Bernot, S. Beroff, D. Dupont, L. Freund, M. Israel, F. Rousseau, *ECOS: Environnement générique de cospécification*, chapitre de l'ouvrage : CODESIGN conception conjointe logiciel-matériel, C.T.I. COMETE, Eyrolles 1998, ISBN 2-2120-5219-7.

[O4] M. Aiguier, J. Benzakki, G. Bernot, S. Beroff, D. Dupont, L. Freund, M. Israel, F. Rousseau, *ECOS: A Generic Codesign Environment for the Prototyping of Real Time Applications "From Formal Specifications to Hardware/Software Partitioning*", chapitre de l'ouvrage : Hardware/Software Co-Design and Co-Verification, Current Issues in Electronic modeling, N° 8, pp. 23 - 57, Kluwer AP, 1997, ISBN 0-7923-9689-8.

[RI2] L. Freund, M. Israel, F. Rousseau, J. M. Berge, M. Auguin, C. Belleudy, G. Gogniat, *A codesign experience in acoustic echo cancellation: GMDFα*, ACM Transactions on Design Automation of Embedded Systems, vol. 2, N. 4, october 1997.

[CI11] L. Freund, D. Dupont, M. Israel, F. Rousseau, *Interface Optimization during Hardware-Software Partitioning*, Workshop on Hardware/Software Co-Design (CODES/CASHES 1997), Brunshweig, Allemagne, mars 1997.

[CI12] L. Freund, M. Israel, F. Rousseau, J-M. Berge, M. Auguin, C. Belleudy, G. Gogniat, *A Codesign Experiment in Acoustic Echo Cancellation: GMDFα*, International Symposium on System Synthesis (ISSS 1996), pp. 83 – 89, La Jolla, Ca, EU, novembre 1996.

[CI13] F. Rousseau, J.-M. Berge, M. Israel, *Hardware/Software Partitioning for Telecommunication Systems*, Computer Software and Application Conference (COMPSAC 1996), pp. 484 – 489, Séoul, Corée du Sud, août 1996.

[CI14] F. Rousseau, J. Benzakki, J.-M. Berge, M. Israel, *Adaptation of Force-Directed Scheduling Algorithm for Hardware/Software Partitioning*, Workshop on Rapid System Prototyping (RSP 1995), pp. 33 – 38, Chapel Hill, NC, USA, juin 1995.

IV Architecture mémoire des systèmes multiprocesseurs monopuces

L'équipe SLS du TIMA s'intéresse à l'architecture mémoire des systèmes multiprocesseurs monopuces depuis oct. 1999. Ce travail a été mené par Samy Meftali et moi-même. Il est inspiré des besoins mémoire des applications multimédias et des difficultés apparues lors de la conception des systèmes avec le flot ROSES.

IV.1 Abondance de mémoires

L'architecture mémoire, dans les systèmes multiprocesseurs monopuces joue un rôle de plus en plus important et retient toute l'attention des concepteurs d'aujourd'hui avec les applications de traitement d'images et de multimédia. Ces applications manipulent des gros volumes de données nécessitant l'intégration d'une mémoire globale partagée. L'architecture mémoire pour de telles applications devient très complexe. Elle occupe jusqu'à 70% de la surface du système et nécessite une grande partie du temps de conception. Elle influe aussi sur les performances de l'architecture. Actuellement, il n'existe pas d'outil permettant l'intégration d'une architecture mémoire générique (distribuée partagée) dans un système multiprocesseur monopuce, à partir d'une spécification système. Donc, le concepteur n'a pas d'autres recours que de restreindre l'espace d'exploration à des architectures mémoire constituées de caches et de mémoires locales, ou de concevoir manuellement l'architecture mémoire. En ciblant une architecture mémoire distribuée partagée, le nombre d'architectures possibles croit d'une façon exponentielle par rapport au nombre de processeurs, et ces différents choix peuvent engendrer des coûts de conception très différents. Ainsi, il est impératif pour le concepteur de faire le choix optimal.

Les avancées technologiques permettent maintenant d'intégrer sur une même puce de silicium différents types de mémoire en plus des composants (les prévisions de l'association de l'industrie des semi-conducteurs et de l'ITRS indiquent qu'en 2014, plus de 90% de la surface de silicium sera occupée par de la mémoire). On trouve des mémoires classiques telles que DRAM (principalement les variantes synchrones SDRAM), les mémoires caches dans les processeurs, mais aussi des mémoires SRAM ("scratchpad", TCM) ou flash. Les mémoires TCM ("tightly coupled memory") sont des mémoires SRAM proches du processeur, accessibles en un cycle dans le cas de l'ARM946E-S. Idem pour les mémoires "scratchpad" qui remplacent la mémoire cache en fixant son contenu. L'intérêt de la mémoire "flash" est de conserver les valeurs même en cas de rupture de l'alimentation.

Pour satisfaire les exigences des applications, les architectures monopuces doivent garantir une grande performance en terme de temps et une faible consommation d'énergie [ABR02]. L'intégration des mémoires sur la puce ne fait que répondre à ces exigences. En effet, le temps d'accès à une mémoire interne à la puce est beaucoup plus court que celui d'accès à une mémoire externe. De plus, la mémoire étant physiquement très proche, on réduit la consommation d'énergie. Ceci ne fait qu'optimiser le temps de calcul en évitant les opérations inutiles, et réduire le temps d'accès global à la mémoire en évitant les protocoles d'accès lents et les transferts de données avec l'extérieur de la puce. Si la mémoire est embarquée sur la puce, le nombre de broches peut également être réduit et l'utilisation de bus sur carte devient obsolète.

IV.2 Architectures mémoire

Les systèmes multiprocesseurs monopuces correspondent à la catégorie des machines MIMD ("Multiple Instructions Multiple Data") selon la classification de Flynn datant de 1966. Chaque processeur lit ses propres instructions et opère sur ses propres données. Une machine MIMD fournit de la flexibilité. Elle peut fonctionner comme une machine mono utilisateur destinée à la haute performance, ou comme une machine multitâche. Une machine MIMD peut être construite en s'appuyant sur les avantages coût - performance des microprocesseurs standard. Les systèmes MIMD actuels sont classables en deux groupes, selon le nombre de processeurs qui lui-même impose une structure mémoire et une stratégie d'interconnexion. On désigne les systèmes selon leur organisation mémoire, car le nombre de processeurs (petit ou grand) a toutes les chances de changer avec le temps. Ainsi, on distingue les systèmes à mémoire partagée et ceux à mémoire distribuée.

IV.2.1 Les systèmes à mémoire partagée

Ces systèmes partagent une mémoire centralisée unique et un bus pour interconnecter les processeurs et la mémoire. Avec de gros caches pour chaque processeur, la mémoire unique peut satisfaire les besoins mémoire d'un petit nombre de processeurs. Avec une seule mémoire principale et un temps d'accès uniforme pour chaque processeur, ces systèmes sont parfois appelés UMA ("Uniform Memory Access").

La facilité et la portabilité de la programmation sur de tels systèmes réduisent considérablement le coût de développement des applications parallèles. Par contre, ces systèmes souffrent d'un handicap qui est la grande latence dans l'accès à la mémoire, ce qui rend la flexibilité (l'extensibilité de l'architecture pour d'autres applications) assez limitée. Ce type d'architectures à mémoire partagée centralisée reste de loin l'organisation la plus populaire.

IV.2.2 Les systèmes à mémoire distribuée

Les systèmes mettant en œuvre une mémoire physiquement distribuée sont souvent appelés "multi-ordinateurs". Ils sont constitués de plusieurs nœuds indépendants et interconnectés. Chaque nœud correspond à un ou plusieurs processeurs et de la mémoire partagée. L'architecture de ces systèmes est dite de type NUMA ("Non Uniform Memory Access"), car en pratique, l'accès à la mémoire locale à un processeur est nettement plus rapide que l'accès à la mémoire locale d'un processeur distant via le réseau de communication.

La nature flexible de tels systèmes, les rend d'une très grande capacité de calcul. Mais, la communication entre des processus résidant dans des nœuds différents nécessite l'utilisation de modèles de communication par passage de messages. En optant pour ce type de systèmes, le concepteur doit particulièrement faire attention à la distribution des données et à la gestion des communications. Les problèmes logiciels, contrairement aux problèmes matériels sont relativement complexes dans les systèmes à mémoire distribuée.

IV.2.3 Les systèmes à mémoire distribuée partagée

Un système à mémoire distribuée partagée (DSM pour "Distributed Shared Memory"), combine les avantages des deux approches précédentes. Un système DSM implante (logiquement) un système à mémoire partagée sur une mémoire physiquement distribuée. Ces systèmes préservent la facilité de programmation et la portabilité des applications sur des systèmes à mémoire distribuée, sans imposer pour autant la gestion des communications par le concepteur. Les systèmes DSM, permettent une modification relativement simple et une exécution efficace des applications déjà existantes sur des systèmes à mémoire partagée, tout en héritant de la flexibilité des systèmes à mémoire distribuée.

Un système multiprocesseur avec mémoire distribuée partagée est généralement constitué d'un ensemble de nœuds (clusters), connectés par un réseau d'interconnexion. Un nœud peut être soit un simple processeur ou une hiérarchie qui cache une autre architecture multiprocesseur, souvent organisée autour d'un bus partagé. Les caches des processeurs sont d'une grande importance afin de réduire la latence. Chaque nœud possède un module de mémoire local (physiquement), faisant partie du système DSM global, ainsi qu'une interface le connectant au système.

La cohérence de la mémoire est sans doute l'un des problèmes majeurs que rencontrent les concepteurs de systèmes multiprocesseurs avec mémoire partagée. En effet, dans une architecture multiprocesseurs, ces derniers communiquent via les données partagées. De ce fait, une question importante se pose : dans quel ordre un processeur doit-il observer les écritures de données d'un autre processeur ? Puisque la seule manière d'observer les écritures des autres processeurs est la lecture, alors quelles propriétés doivent être respectées entre les lectures et les écritures dans les emplacements mémoire par les différents processeurs ? Il existe plusieurs modèles de cohérence mémoire (séquentielle, par invalidation, par diffusion, …).

IV.2.4 Notre choix de systèmes à mémoire distribuée partagée

Nous nous sommes principalement intéressé aux mémoires de données et donc à la conception des systèmes à mémoire distribuée partagée. On distingue trois types de mémoire : mémoire locale privée (mémoire de petite taille, non accessible directement par les autres processeurs), mémoire locale distribuée (mémoire directement accessible par les autres processeurs) et une mémoire globale partagée (mémoire de grande taille accessible de façon uniforme par plusieurs processeurs).

Le problème de la cohérence mémoire est un problème crucial auquel il faut sans doute accorder une grande importance lors de la conception de systèmes multiprocesseurs utilisant des mémoires distribuées partagées. Pour simplifier notre démarche, nous avons pris certaines hypothèses présentées ci-dessous avec leur justification.

- La spécification de haut niveau contient explicitement la synchronisation, et gère la cohérence. La description d'une application sous forme d'un ensemble de tâches (en SDL par exemple), requiert toutes les informations en vue de la simulation. C'est donc au concepteur de les spécifier.
- L'allocation dynamique de mémoire n'est pas supportée, ce qui permet de prévoir et d'éviter tout problème de cohérence par le concepteur dès les hauts niveaux d'abstraction.
- Les caches sont utilisés pour améliorer les performances d'une architecture généraliste pour une application. Le fait de tailler l'architecture pour une application donnée pourrait nous permettre de s'affranchir des caches et ils peuvent aussi être avantageusement remplacés par des mémoires "scratchpad", ayant les mêmes caractéristiques temporelles mais une gestion plus simple et déterministe. Néanmoins, les caches étant présents avec les processeurs, une politique d'écriture simultanée a le mérite de simplifier la gestion de la cohérence, au détriment il est vrai de la performance (augmentation de la charge du bus vers la mémoire).

IV.3 Méthode d'exploration d'architectures et allocation/affectation mémoire

IV.3.1 Objectifs

Les objectifs des travaux de thèse de Samy Meftali [MEF02b] sont de définir et d'intégrer un flot automatisable de conception d'une architecture mémoire dans le flot ROSES. Ce flot doit permettre de choisir une architecture mémoire, de fixer le nombre, les emplacements et la taille des différents

blocs mémoire dans le système. L'affectation mémoire est l'opération qui attribue à chaque donnée de l'application un emplacement dans l'un des blocs mémoire alloués.

Il faut donc dans un premier temps reconnaître dans le modèle de haut niveau (spécification comportementale) les éléments à mémoriser et en déduire l'architecture mémoire la mieux adaptée à l'application, et la taille des composants mémoire. Ceci entraîne une modification de l'architecture du système en COLIF et une modification du code applicatif (ou d'E/S) pour tenir compte de cette nouvelle architecture mémoire. Il faut aussi affecter les données de l'application dans les différentes mémoires de façon optimale.

IV.3.2 Etat de l'art de l'allocation/affectation mémoire

A notre connaissance, il n'existe pas à ce jour d'outils industriels permettant de concevoir automatiquement une architecture optimisée. Ce problème est abordé par quelques travaux de recherche mais reste encore assez marginal. En effet, les flots de conception existants dans la littérature se concentrent tous sur l'automatisation de la conception de la partie communication du système, ainsi que sur la connexion des différents processeurs entre eux et/ou avec des parties matérielles spécifiques. Mais en aucun cas ils n'envisagent l'intégration d'une architecture mémoire à partir d'un haut niveau d'abstraction. En effet la mémoire n'est intégrée dans le système qu'à un niveau d'abstraction relativement bas.

De nombreux travaux existent sur l'optimisation de la taille des caches, en extrayant par exemple des paramètres des programmes sources pour analyser puis synthétiser un cache optimal. Ce sont des approches d'exploration plus ou moins exhaustives de différentes configurations de caches. Ces approches sont très peu conseillées pour des applications manipulant des données volumineuses, car dans ce cas on voit apparaître des problèmes de défauts de caches liés essentiellement aux petites tailles des caches.

L'université d'Irvine s'intéresse aux problèmes des caches et est particulièrement avancée et représentative de ce type d'exploration d'architectures mémoires spécifiques [PAN99][GRU03][MIS04]. Ils se basent sur l'exploration de différentes configurations de caches, mais aussi sur l'utilisation des mémoires particulières à accès très rapide telles que les "scratchpad" [Gru00]. Des travaux ont été menés en collaboration entre cette université et l'équipe SLS sur l'intégration d'un coprocesseur dans le cœur d'un processeur VLIW [MIS01a][MIS01b], ce qui suppose de bien réfléchir à la gestion des communications internes et de la mémoire du processeur et à la génération du compilateur et du simulateur pour tenir compte de ces caractéristiques.

Certains travaux dans la littérature traitent le problème de conception de l'architecture mémoire pour certains types d'applications d'une façon plus au moins complète, en essayant de mixer les niveaux de caches avec des mémoires internes et externes. Les plus connus de ces travaux sont sans doute ceux de l'IMEC, et qui ont permis d'élaborer la méthode DTSE ("Data Transfer and Storage Exploration") [CAT98][CAT02]. Cette approche est destinée à des applications temps réel orientées flot de données très spécifiques telles que certaines parties des applications multimédia (audio et vidéo). Son but est d'optimiser essentiellement deux critères, la surface et la consommation, d'une grande importance pour ce type d'application. Partant d'un graphe représentant le transfert des données et les conflits d'accès, la méthode DTSE a pour objectif de trouver une bonne architecture mémoire (taille, nombre de mémoires, connexions,..). Cette méthode semble efficace et elle réalise une allocation mémoire et affectation des données de l'application en cherchant à minimiser le nombre d'accès mémoire et la taille des mémoires. Mais ceci entraîne des modifications dans le code applicatif, qui risque d'augmenter le nombre de cycles. De plus, cette méthode ne semble pas prendre en compte les mémoires distribuées partagées.

IV.3.3 Description de notre flot de conception de l'architecture mémoire

Notre flot de conception (Figure 7) accepte en entrée une spécification de l'application au niveau système. Elle décrit le comportement de l'application, c'est-à-dire le comportement des différentes fonctions composant le système, les échanges de données entre ces fonctions ainsi que l'utilisation des canaux de communication par le flux de données échangées.

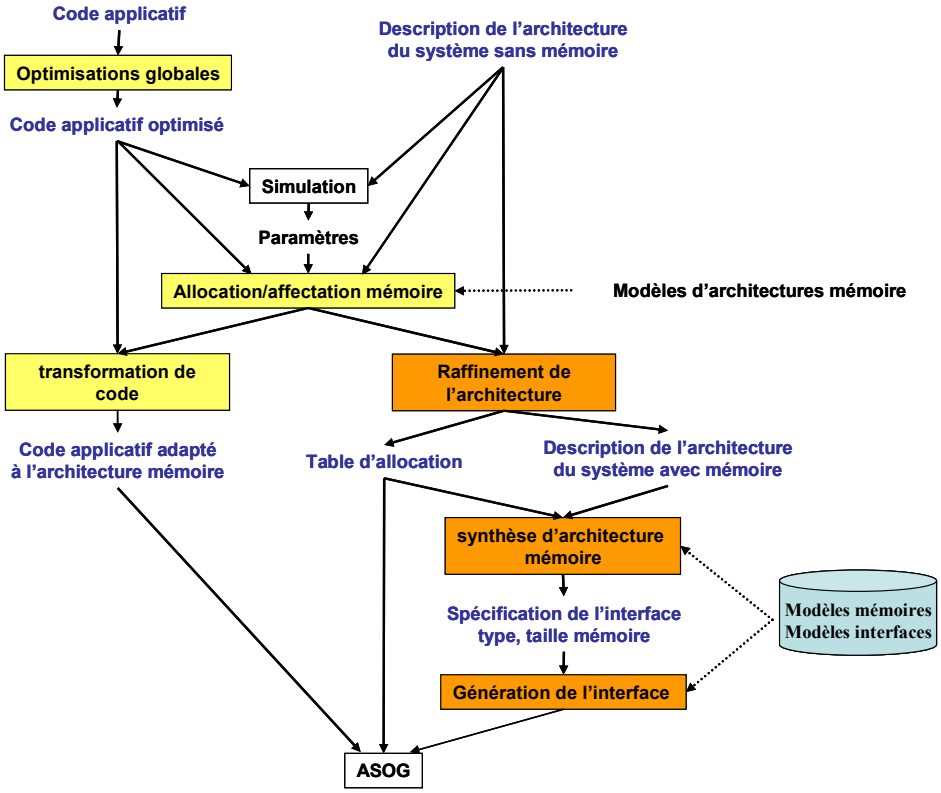

Figure 7 : Flot de conception de l'architecture mémoire

Une première étape d'optimisation globale transforme le code applicatif initial, dans le but de diminuer le nombre d'accès en lecture et en écriture des variables. On cherche alors à réduire la distance d'accès à une variable ou à changer l'ordre de parcours des éléments d'un tableau [FRA99][FRA01]. Ces transformations sont indépendantes de l'architecture physique.

La simulation valide les transformations précédentes, et permet d'obtenir alors des informations sur le partage des données entre les différents blocs du système ainsi que sur l'utilisation des canaux de communication. Ces informations sont utilisées dans la suite du flot et plus particulièrement lors de l'allocation mémoire.

Le but de l'allocation est d'allouer tous les blocs mémoire pour l'application (mémoires locales privées, locales partagées, et globale partagée). Cette allocation est déterminante pour les performances du système final (surface, consommation en énergie, performances temporelles,…). Ainsi, l'allocation mémoire a pour objectifs d'allouer des blocs mémoires d'une façon optimale pour l'application, et de déterminer quel bloc mémoire est le mieux adapté au stockage de chacune des données. Un modèle basé sur la programmation linéaire en nombres entiers [MEF01] a été développé pour réaliser cette allocation mémoire de façon optimale en minimisant le temps global d'accès aux variables partagées du système (variables globales et variables de communication).

L'étape d'allocation implique la modification du code du modèle de l'architecture (plusieurs dizaines de fichiers) et la modification du code applicatif (comportement des tâches) s'exécutant sur les processeurs, la génération de code (éventuellement du bloc mémoire globale partagée) ainsi que l'exécution de plusieurs algorithmes d'optimisation. Malgré cette complexité, l'étape du

raffinement et transformation de code (adaptation du code de l'application à l'architecture mémoire allouée) est assez systématique, ce qui rend son automatisation très bénéfique car elle peut réduire considérablement le temps de conception et les erreurs [Mef02a].

L'étape d'affectation consiste à distribuer l'ensemble des données de l'application sur les mémoires allouées au cours de l'allocation mémoire. Ainsi, à l'issue de cette étape d'affectation chaque donnée de l'application doit posséder une adresse physique dans une mémoire. Elle est faite simplement sans algorithme d'optimisation, mais il existe des "bonnes solutions" basées sur des algorithmes gloutons ou sur le problème classique du "Bin-Packing" qui consiste à ranger des objets de tailles connues dans des espaces de tailles limitées en optimisant l'espace utilisé.

Quand les données de l'application sont affectées aux différents espaces adressables, un fichier intermédiaire (table d'allocation) est généré. Il contient toutes les informations sur les emplacements des variables. Ainsi, pour chaque donnée, on connaît la mémoire dans laquelle elle réside et son adresse. Cette table d'allocation est utile lors de la génération du système d'exploitation spécifique par ASOG.

La dernière étape consiste à produire physiquement les composants mémoire (réutilisation de composants existants, synthèse, …). Des adaptateurs mémoire sont nécessaires pour rendre compatibles les protocoles et les liens physiques entre un composant mémoire et le media de communication auquel il est relié. Cet adaptateur joue aussi le rôle de contrôleur. Ces adaptateurs mémoire sont détaillés dans le chapitre suivant.

IV.3.4 Bilan et limites de la méthode

Le flot d'allocation mémoire et le raffinement présentés dans ce chapitre ont été réalisés en C. La programmation se compose essentiellement des trois modules. L'analyseur de code extrait toutes les informations nécessaires à la génération du programme linéaire en nombres entiers, en parcourant principalement les fichiers du code applicatif et de la description de l'architecture du système. Le second module est le générateur et le "solveur" du programme linéaire en nombres entiers. Il fait appel aux bibliothèques Cplex de Ilog pour résoudre le programme et obtenir ainsi l'architecture mémoire, et une affectation abstraite des données aux différentes mémoires. Le troisième module se charge de la génération du bloc mémoire partagée au niveau architecture, des contrôleurs et de la transformation des primitives d'accès aux données partagées dans la spécification initiale de l'application. Il constitue le module le plus fastidieux à programmer surtout dans le cas où les primitives de communication sont cachées dans le reste du code de l'application. Ce module est partiellement automatisé et il nécessite encore une intervention manuelle sur le code de l'application.

Il existe plusieurs limitations à cette méthode. Le fait d'extraire les paramètres à partir d'un résultat de simulation pour l'allocation suppose que le vecteur de test est représentatif du comportement et considère un fonctionnement normal (valeurs min, typique, max ?). Toutefois, si certaines informations sur les accès aux données sont connues statiquement, de nombreuses informations sont connues seulement dynamiquement, et une estimation précise semble difficile.

Mais la principale limitation concerne le modèle et les coefficients de la fonction objectif du programme linéaire en nombres entiers. Ce modèle fait appel à plusieurs valeurs telles que le temps d'accès en lecture et en écriture d'un processeur à une mémoire locale et à une mémoire partagée. Si le temps d'accès à la mémoire locale est généralement bien connu, l'accès à une mémoire partagée est fortement dépendant du reste du système (charge du bus, bande passante, protocole de communication, ordonnancement des tâches, traitement des E/S, …) et donc connu dans un intervalle de valeurs. Les équations du modèle utilise aussi comme coefficient le coût de l'architecture mémoire. C'est la somme de deux termes, l'un dépendant de la taille de chaque mémoire (et proportionnel à la taille des données affectées), l'autre correspond à la fois à l'effort de

l'intégration en tant que composant et au coût financier engendré par cette intégration. Ces valeurs sont difficilement chiffrables en valeur absolue !

Enfin, étant donné que les variables de ce modèle linéaire sont de type binaire, sa résolution peut être très lente en fonction du nombre de variables. Ainsi il est souhaitable de n'utiliser dans ce modèle que les variables importantes de l'application, c'est-à-dire les plus volumineuses ou celles auxquelles le processeur accède le plus souvent.

IV.4 Conclusion et perspectives

La méthode d'allocation et d'affectation présentée dans le paragraphe précédent nous a permis de comprendre l'importance de l'architecture mémoire et son influence sur les performances. Trop de paramètres interviennent dans le système pour facilement dire que l'architecture mémoire est optimale, surtout quand le système est dynamique ou fortement dépendant des données. De plus, dans notre étude, les caches n'ont un pas vraiment été pris en compte. La raison principale est qu'au début de cette étude, un des processeurs embarqués le plus couramment utilisé ne possédait pas de caches (certaines versions de l'ARM7 TDMI). De plus, on pensait alors que les architectures de systèmes étant dédiées à une application, ceci permettait de s'affranchir du problème des caches. Maintenant, tous les processeurs ont une mémoire cache (deux pour la plupart) et les architectures sont pensées en terme de famille d'applications, ce qui remet en cause une partie de nos hypothèses de départ, et rend donc obsolètes ces travaux dans la version présentée. Une solution simple est possible, en ne tenant pas compte des caches pendant l'allocation/affectation et en acceptant que les caches améliorent les performances pendant l'exécution. Une autre solution est de modifier les hypothèses pour tenir compte des caractéristiques des mémoires cache, ce qui nécessite une étude et des travaux plus importants.

IV.5 Principales publications relatives au chapitre IV

[O1] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya, *Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, chapitre de l'ouvrage : SoC Design Methodologies, Kluwer Academic Publishers, juin 2002, ISBN 1-4020-7148-5.

[CI7] S. Meftali, F. Gharsalli, F. Rousseau, A. A. Jerraya, *Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, International Conference on Very Large Scale Integration: the global System on Chip Design (VLSI-SoC 2001), pp. 17-22, Montpellier, France, déc. 2001.

[CI9] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya, *An Optimal Memory Allocation for Application-Specific Multiprocessor System-on-Chip*, International Symposium on System Synthesis (ISSS 2001), pp. 19-24, Montréal, Canada, sept-oct. 2001.

IV.6 Encadrement de thèse

Samy MEFTALI. *Exploration d'architectures et allocation/affectation mémoire dans les systèmes multiprocesseurs monopuces*. Doctorat de l'Université Joseph Fourier – Grenoble soutenue le 6 sept. 2002. En co-direction avec Ahmed Jerraya. Taux d'encadrement de 80%.

V Conception automatique des interfaces de communication

Les travaux sur l'architecture mémoire ont fait apparaître le besoin d'interfaces logicielles et matérielles pour exploiter pleinement les performances de l'architecture mémoire. Cet axe de recherche a commencé avec les travaux de DEA et de thèse de Ferid Gharsalli en avril 2000 et se poursuit avec les travaux d'Arnaud Grasset.

V.1 Problématique

V.1.1 Nécessité des interfaces logiciel/matériel

La conception des systèmes multiprocesseurs monopuces est faite par assemblage de composants existants autour d'un réseau de communication pour des raisons de rapidité de conception. Et pour répondre spécifiquement aux besoins de l'application, les composants utilisés sont de nature différente. Pour ces deux raisons, il est nécessaire de placer une interface de communication entre chaque composant et le réseau de communication pour adapter physiquement les connexions entre ces éléments et pour rendre compatible les protocoles de communication mis en œuvre (Figure 1). Ces composants d'interface de communication, appelés aussi interfaces matérielles, sont généralement pilotés par un processeur maître à travers une fonction d'entrée/sortie (au sens de programme) appelée pilote (ou "driver"). Ce pilote logiciel et ce composant d'interface matérielle correspondent aux interfaces logiciel/matériel.

Pour rendre efficace la conception de systèmes multiprocesseurs monopuces à base de réutilisation de composants existants, le concepteur doit consacrer beaucoup d'efforts pour la spécification, l'implémentation et la validation des interfaces logiciel/matériel, ce qui devient un réel problème de conception.

La conception des interfaces logiciel/matériel pour les processeurs a été traitée dans les thèses de Damien Lyonnard [LYO03] et Lovic Gauthier [GAU01]. On s'intéresse dans la suite de ce chapitre aux interfaces logiciel/matériel pour les composants mémoires. A partir des critiques de la méthode de génération, on essaie de proposer une nouvelle approche dont les travaux sont toujours en cours pour les étendre à toutes les interfaces de communication.

Les travaux décrits dans la suite de ce chapitre se limitent aux mémoires globales (partagées ou non), mais n'abordent pas le cas des mémoires locales, directement connectées aux bus du processeur. L'accès par un processeur à la mémoire locale d'un autre processeur (mémoire globale distribuée) amène une autre problématique qui n'est pas traitée.

V.1.2 Difficulté de conception des interfaces logiciel/matériel pour les mémoires

Un système monopuce contient généralement plusieurs composants mémoires connectés au réseau de communication. Un composant d'interface matériel s'insère entre le composant mémoire et le réseau de communication pour adapter les protocoles de communication, pour générer les signaux physiques nécessaires au bon fonctionnement de la mémoire et effectuer des transformations sur les données selon la nature et les caractéristiques des informations transitant sur le réseau et celles acceptées par le composant mémoire.

La principale difficulté liée à la conception des interfaces matérielles vient de la très grande variété des composants mémoires et des réseaux de communication. Les protocoles d'accès mémoire se diversifient de plus en plus, ce qui rend plus difficile leur adaptation aux réseaux de communication, notamment pour profiter de tous les protocoles disponibles (mode rafale, …).

Les réseaux de communication étant devenus de plus en plus complexes, la spécification et la validation de la communication au niveau transfert de registre (RTL) sont plus difficiles. En effet, à ce niveau d'abstraction, la structure d'un média de communication (bus, réseau) est détaillée au niveau du cycle pour vérifier les contraintes logiques et électriques du système. Ce niveau est trop bas pour la conception de la communication qui correspond dans notre cas essentiellement à la conception des interfaces logiciel/matériel. Pour ces raisons, un niveau d'abstraction plus élevé que le niveau RTL est nécessaire pour la spécification et la validation des interconnexions entre les composants d'un système monopuce.

Généralement, le code de l'application est décrit à un haut niveau d'abstraction indépendamment de l'architecture mémoire. Le code de l'application n'accède pas directement aux éléments de mémorisation, mais utilise l'interface de communication logicielle appelé aussi "pilote d'accès". Ce pilote est une fonction qui cache les caractéristiques matérielles, rendant le code de l'application indépendant de l'architecture. Dans de nombreux systèmes, ces pilotes sont gérés par le compilateur du processeur. Dans un système embarqué, des pilotes spécifiques sont nécessaires, d'une part pour obtenir un code de taille réduite, d'autre part pour s'adapter aux spécificités de l'architecture du système (décodage et translation d'adresses pour les mémoires globales, pilotage des interfaces matérielles, gestion des signaux de communications, …).

Cette interface logicielle facilite la portabilité du code de l'application sur différents types de processeurs. Cette portabilité facilite aussi la réutilisation du logiciel pour accélérer le processus de conception et réduire le temps de mise sur le marché. L'interface logicielle peut également être vu comme une couche de séparation entre le monde logiciel et le monde matériel, ce qui permet une conception concurrente.

L'aspect logiciel des problèmes d'intégration mémoire est lié à la nécessité des pilotes logiciels qui font correspondre les accès mémoire du code de l'application aux processeurs sur lesquels les tâches de l'application sont exécutées. Le développement de ces pilotes est une tâche difficile, causée essentiellement par leur forte dépendance au matériel car ils sont très liés à l'architecture du processeur sur lequel ils s'exécutent (taille du bus de données, taille du bus d'adresses, le mode d'adressage, type de données transférées, etc.). Cette forte dépendance ne fait que réduire leur flexibilité. Pour les rendre moins dépendants du matériel et donc réutilisables, une implémentation modulaire et en couches s'impose.

V.1.3 Etat de l'art sur la conception des interfaces matérielles pour les mémoires

On distingue deux types d'interfaces matérielles selon que le protocole de communication est spécifique ou standard.

La conception des interfaces matérielles spécifiques consiste à raffiner une spécification jusqu'à obtenir une réalisation finale spécifique à un protocole de communication donné. Le principe commun de toutes ces approches est le raffinement continu des interfaces et des protocoles de communication [VER96][JEN97][KNU98][GAJ98][ISM96]. Ce raffinement aboutit à une réalisation spécifique dont l'interface ne correspond à aucun standard. L'automatisation de ces méthodes de conception repose essentiellement sur des bibliothèques logicielles et matérielles implémentant les unités de communication spécifiques. L'inconvénient de ces approches est qu'elles ne supportent que leurs protocoles de communication internes, mais n'acceptent pas ceux définis à l'extérieur de leur environnement.

La conception des interfaces matérielles standard consiste à utiliser des protocoles de communication et des interfaces qui suivent un standard donné. Récemment, beaucoup de travaux de recherche se sont intéressés à la standardisation des protocoles de communication [BOR98] [ONI01]. Le but de ces travaux est de faciliter la réutilisation des composants. Le concept de base de ces approches est de standardiser les interfaces des bus de communication. Un tel standard permet au concepteur d'IP de supporter un nombre limité d'interfaces.

Parmi les autres travaux de standardisation les plus connus, on trouve l'approche de Virtual Socket Interface Alliance (VSIA) qui définit un standard d'interfaces de bus nommé Virtual Component Interface (VCI) [BRU00].

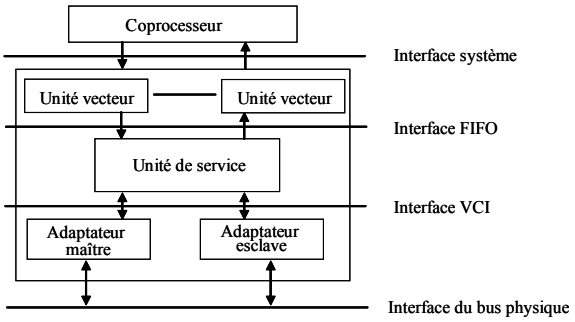

Figure 8 : Interface matérielle utilisant le standard VCI

La Figure 8 montre la structure d'une interface de communication matérielle utilisant la norme VCI. Cette interface permet à un coprocesseur (ou à un composant mémoire) de s'adapter à plusieurs bus en utilisant le standard VCI-OCB1. Cette interface est découpée en trois parties. L'interface système est définie comme des unités de vecteurs qui transforment les appels systèmes en un simple protocole producteur-consommateur implémenté par des FIFO. L'unité de service transforme le protocole FIFO en un protocole VCI. Cette unité fournit des services de contrôle pour les FIFO et des services de configuration du matériel. L'adaptateur du bus adapte l'interface VCI à l'interface du bus de communication.

Il existe d'autres efforts de standardisation : PIBus, CoreConnect, OpenCore, … . Bien que les approches de standardisation facilitent l'assemblage de composants, elles connaissent aujourd'hui plusieurs problèmes :

- Les protocoles standard ne répondent pas toujours aux exigences spécifiques d'une application.
- Les interfaces de communication standard sont parfois trop générales et utilisent des protocoles non nécessaires.

V.1.4 Etat de l'art sur la conception des interfaces logicielles pour les mémoires globales

En plus de la nécessité de l'adaptation matérielle, l'intégration d'une mémoire globale dans un système monopuce nécessite une adaptation de type logiciel. Ces pilotes d'accès assurent la portabilité du code d'accès mémoire sur différentes architectures utilisant des processeurs et des systèmes d'exploitation différents. Ces pilotes sont très liés au matériel, et les compilateurs savent les générer uniquement dans le cas d'architectures standard (pilotages de cartes graphiques, cartes audio). Plusieurs outils commerciaux de développement logiciel fournissent les pilotes classiques aux processeurs à usage général. Ces pilotes sont trop généraux pour être utilisés par les systèmes monopuces. En effet, ils ne sont pas conçus spécifiquement à l'application et ils peuvent implémenter des fonctionnalités non nécessaires. Récemment, plusieurs travaux de recherche ont été menés pour trouver de nouvelles méthodes de conception de pilotes monopuces [BOR98] [KAT99]. [I2O02] présente une architecture de pilote logiciel plus flexible composée de trois modules superposés :

- Un module qui dépend du système d'exploitation. Ce module est aussi appelé OSM ("Operating system Service Module").

[1] VCI-OCB (Virtual Component Interface-On Chip Bus) : c'est une spécification qui définit les interfaces et les protocoles d'un bus de communication monopuce. Cette spécification a été développée par le groupe OCB de VSIA.

- Un module dépendant du matériel. Ce module est généralement appelé HDM ("Hardware Device Module").
- Un média de communication entre les deux autres modules appelés ISM ("Intermediate Service Module"). Cette couche de communication est basée sur un protocole d'envoi de message ("message passing").

Cette conception modulaire sert à fournir une interface standard entre les systèmes d'exploitation et les pilotes de périphériques. La limite de cette approche est qu'elle n'apporte aucune réalisation.

V.1.5 Objectifs et point de départ dans ROSES

Une méthode de génération d'interfaces matérielles ouvertes aux protocoles de communication spécifiques ou standard a été développée dans l'équipe SLS et a abouti à un outil de génération d'interfaces (ASAG) basé sur l'assemblage de composants de bibliothèque. Il permet la génération d'interfaces matérielles de communication capables d'adapter un processeur à un réseau de communication. Mais la génération des interfaces matérielles pour les mémoires n'avait pas été étudiée.

L'outil ASOG est un outil de génération de système d'exploitation. Il est aussi basé sur l'assemblage de composants logiciels issus d'une bibliothèque. Il devrait permettre la génération des pilotes pour la mémoire.

Ces deux outils offrent à la fois un outil de composition et une bonne flexibilité pour étendre les travaux à un domaine plus large. Dans notre cas, il s'agit de les étendre dans un premier temps vers les composants mémoire.

Mais ce travail sur les interfaces logiciel/matériel pour la mémoire suppose de réfléchir à d'autres aspects : Comment spécifier à un plus haut niveau d'abstraction que le niveau RTL ces interfaces ? Comment prendre en compte la grande diversité des composants mémoire et des réseaux de communication ? Comment générer automatiquement les pilotes logiciels et les interfaces matérielles et comment les valider rapidement ? Tout ceci doit s'intégrer avec le flot ROSES, en utilisant au maximum les outils ou éléments de bibliothèque déjà existants.

V.2 Architecture générique des interfaces matérielles

Il existe quelques différences fondamentales entre une interface matérielle processeur et une interface matérielle pour les mémoires. Un processeur est toujours maître vis-à-vis de l'interface qui le sépare du réseau, alors que l'interface mémoire doit être maître pour piloter le composant mémoire. De plus, selon le type de composant mémoire, il faut non seulement générer les signaux d'accès et de protocoles, mais aussi des signaux de fonctionnement spécifiques, le rafraîchissement des SDRAM par exemple, ou la gestion de l'ordre des accès.

Les travaux sur les interfaces matérielles pour les processeurs ont montré l'intérêt d'une approche en trois parties. Une partie est spécifique au réseau de communication. Une seconde partie est spécifique au processeur, et la troisième partie est un bus interne standard qui connecte les deux premières. Ce choix conceptuel est motivé par les besoins de la séparation entre l'interface du média de communication externe et celle de la mémoire. Une telle séparation assure la flexibilité d'utilisation et la réutilisation de ces parties.

Ferid Gharsalli a proposé une architecture d'interfaces mémoire matérielles générique et flexible. Comme l'indique la Figure 9, cette architecture est composée de trois parties essentielles :

- Une partie spécifique aux ports d'accès mémoire appelée adaptateur de port mémoire (MPA pour "Memory Port Adaptor"). Cette partie est spécifique au composant mémoire. Elle

assure l'adaptation d'interface entre la mémoire et le bus de communication interne. Sa fonctionnalité principale est la préparation des adresses et données pour la lecture ou l'écriture, et au transfert des données selon le protocole d'accès mémoire. En effet, selon la gestion des adresses (virtuelle, présence d'une MMU), une traduction d'adresses est nécessaire. Cette partie résout aussi l'hétérogénéité entre le protocole de communication du bus interne et celui de transfert des données de (ou vers) la mémoire.

- Une partie spécifique au média de communication externe appelée adaptateur de canal (CA pour "Channel Adaptor"). Elle adapte l'interface du réseau externe à l'interface du réseau interne. Ici aussi, l'adaptation de protocole, la conversion des données, la gestion de l'ordre d'accès requiert un découpage en plusieurs couches de cette partie.
- Un bus de communication interne à l'adaptateur matériel qui connecte les CA et les MPA. C'est la frontière entre l'interface de la communication et celle de la mémoire. Elle assure l'acheminement des données entre les deux couches. Un arbitre sur ce bus contrôle les accès concurrents.

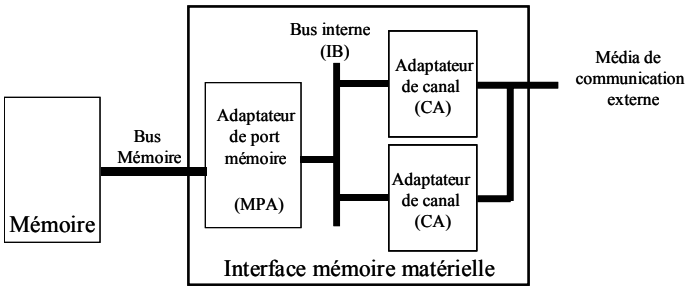

Figure 9 : Architecture interne d'une interface mémoire matérielle

Nous avons opté pour cette décomposition architecturale pour plusieurs raisons :

- Séparation entre le comportement et la communication : les adaptateurs de canaux (CA) contrôlent les protocoles de communication tandis que les adaptateurs de ports mémoire (MPA) gèrent les transferts de données spécifiques vers (ou de) la mémoire.
- Flexibilité : le fait que l'interface mémoire contienne une partie spécifique à la mémoire et une autre spécifique au réseau de communication rend l'exploration architecturale de plusieurs types de mémoire et de réseaux de communication moins complexe. Si le concepteur veut utiliser un nouveau type de mémoire, il suffit de changer le module d'adaptation spécifique à la mémoire (MPA). Les adaptateurs de canaux restent les mêmes.
- Standard d'adaptation : l'utilisation d'un bus interne à l'interface mémoire matérielle assure le découplage entre les interfaces des modules spécifiques à la communication et celles des modules spécifiques à la mémoire. Ce découplage permet d'adapter des protocoles de communication de nature différente à des composants mémoire venant de différents fabricants.
- Facilité de réalisation et de validation : une décomposition modulaire facilite la réalisation ainsi que la validation de ces interfaces matérielles. En effet, on peut réaliser et tester le comportement de chaque module séparément sans se soucier des autres fonctionnalités. Ceci permet de réduire le temps de développement et d'avoir un code facile à maintenir.
- Réutilisation : la décomposition modulaire facilite la réutilisation des adaptateurs de canaux et des adaptateurs de ports mémoire dans plusieurs autres applications.

V.3 Génération automatique des interfaces matérielles

V.3.1 Objectif et résultats attendus

L'objectif de la génération automatique des interfaces mémoire matérielles est de faciliter l'intégration de composants mémoire dans l'architecture d'un système monopuce. La génération automatique de ces interfaces accélère la phase de conception ou le prototypage, et elle facilite aussi l'exploration de l'architecture mémoire.

On attend des interfaces les caractéristiques suivantes : un comportement correct, être synthétisable, d'une complexité optimale, un surcoût en temps de communication acceptable et le temps de génération assez court.

V.3.2 Description de l'outil de génération

L'outil de génération [LYO03] [GHA03] produit le code RTL des interfaces permettant de connecter les processeurs et les mémoires au réseau de communication. La fonctionnalité de l'outil consiste à assembler les composants de base issus d'une bibliothèque. L'outil de génération accepte en entrée trois éléments (Figure 10) :

- Une architecture virtuelle annotée : c'est une spécification (SystemC [SYS00]) logiciel/matériel de l'application annotée avec des paramètres de configuration.
- Un modèle d'architecture du nœud à intégrer, appelé aussi architecture interne, au format intermédiaire (Colif) [CES01]. Pour permettre à l'utilisateur de concevoir ses propres architectures internes, sans être obliger de connaître le format intermédiaire, un outil d'aide à la construction de modèles Colif a été développé.
- Une bibliothèque de comportement : nous avons développé une bibliothèque matérielle contenant des macro modèles de mémoires, d'adaptateurs de canaux (CA) et d'adaptateurs de ports mémoire (MPA).

La construction de l'architecture mémoire interne résulte soit des paramètres entrés par l'utilisateur (nombre de modules, nom et type des modules, nombre, nom et direction des ports, type et taille des données de chaque port, interconnexion entre les modules, liens vers les fichiers source de comportements, …), soit des paramètres obtenus par les travaux décrits au chapitre précédent.

La sortie de l'outil de génération est le code des interfaces matérielles en SystemC ou en VHDL.

La génération automatique des interfaces matérielles est composée de cinq étapes essentielles :

- L'étape de "lecture de la spécification" consiste à extraire les paramètres de configuration qui annotent la spécification d'entrée.
- L'étape de "lecture de l'architecture interne" consiste à charger la description Colif de l'architecture interne de la mémoire. Cette lecture permet de connaître la disponibilité des ressources architecturales nécessaires pour la réalisation de l'architecture cible.
- Le raffinement d'un nœud consiste à définir la structure de l'interface en fixant le nombre et la nature des adaptateurs de canaux (CA) et des adaptateurs de ports mémoire (MPA).
- Le raffinement d'un canal virtuel correspond à une analyse de tous les ports liés à un canal de communication. On en déduit une liste d'attributs caractérisant ce canal. Un modèle de canal parmi les différents modèles disponibles dans la bibliothèque est choisi.
- A chaque instance d'élément d'adaptation (CA, MPA) correspond un modèle d'implémentation dans une bibliothèque générique décrite dans un macro langage.

L'expansion de ce code avec les paramètres de configuration génère le code final de l'interface matérielle qui est simulable et synthétisable.

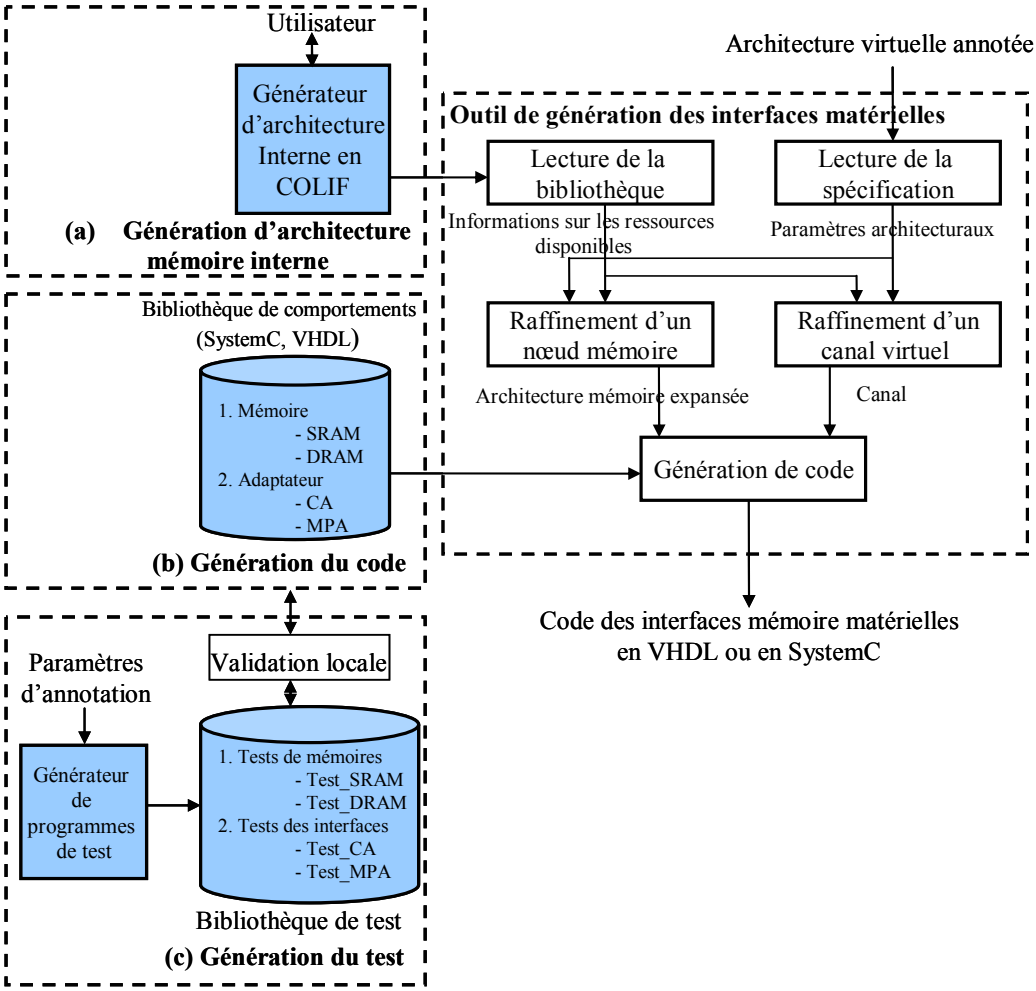

Figure 10 : Environnement de génération des interfaces matérielles

Pour valider les interactions entre l'interface d'un composant et toutes les interfaces possibles des autres composants, nous proposons une méthode de génération automatique de programmes de tests d'interfaces. Chaque test généré est spécifique à une interface de communication donnée. Cette méthode permet de vérifier les interfaces localement avant la validation globale du système. La validation globale de tout le système est basée sur une approche de co-simulation [NIC02]. L'aspect automatique de cette méthode permet au concepteur de couvrir une partie des tests dans un temps minimal.

La bibliothèque matérielle contenant le comportement des macro modèles de mémoires, d'adaptateurs de canaux et d'adaptateurs de ports mémoire est validée en utilisant des programmes de tests génériques. A chaque composant de la bibliothèque, est associé un modèle de programme de simulation SystemC.

V.3.3 Eléments de bibliothèque

La bibliothèque de comportements contient deux types d'éléments : les macro modèles de mémoires et les éléments d'interfaces.

Un macro modèle de mémoire est une implémentation générique de mémoire configurable avec des paramètres qui dépendent généralement de l'application. Un macro modèle n'est ni simulable, ni synthétisable car plusieurs caractéristiques mémoires sont encore abstraites. L'écriture de cette

bibliothèque de macro modèles mémoire dépend du langage cible. Les langages que nous avons visés sont SystemC et VHDL. Le langage d'écriture de cette bibliothèque (RIVE) est indépendant du langage cible. La configuration de ces macro modèles génère des mémoires en SystemC ou en VHDL qui sont simulables et synthétisables. La génération de ce code est obtenue en remplacement systématiquement des paramètres génériques du macro modèle par des valeurs spécifiques à une application donnée.

Les macro modèles de MPA et de CA sont aussi des modèles génériques décrits pour différents types de réseau, de composants mémoire et de protocoles. Pour une application donnée, un lien vers les sources du comportement souhaité et des paramètres de configuration (type de données, taille des bus, …) permet de générer le code VHDL ou SystemC de l'interface.

V.3.4 Exemple de génération

En utilisant le flot décrit précédemment, pour une application de traitement d'images biprocesseurs partageant une mémoire globale (SRAM double port ou SDRAM simple port), on obtient les architectures données sur la Figure 11. En changeant le type de composant mémoire, on constate que la partie CA reste identique. Le MPA est évidemment spécifique et adapté au composant mémoire. Le bus interne est différent selon les accès des processeurs à la mémoire.

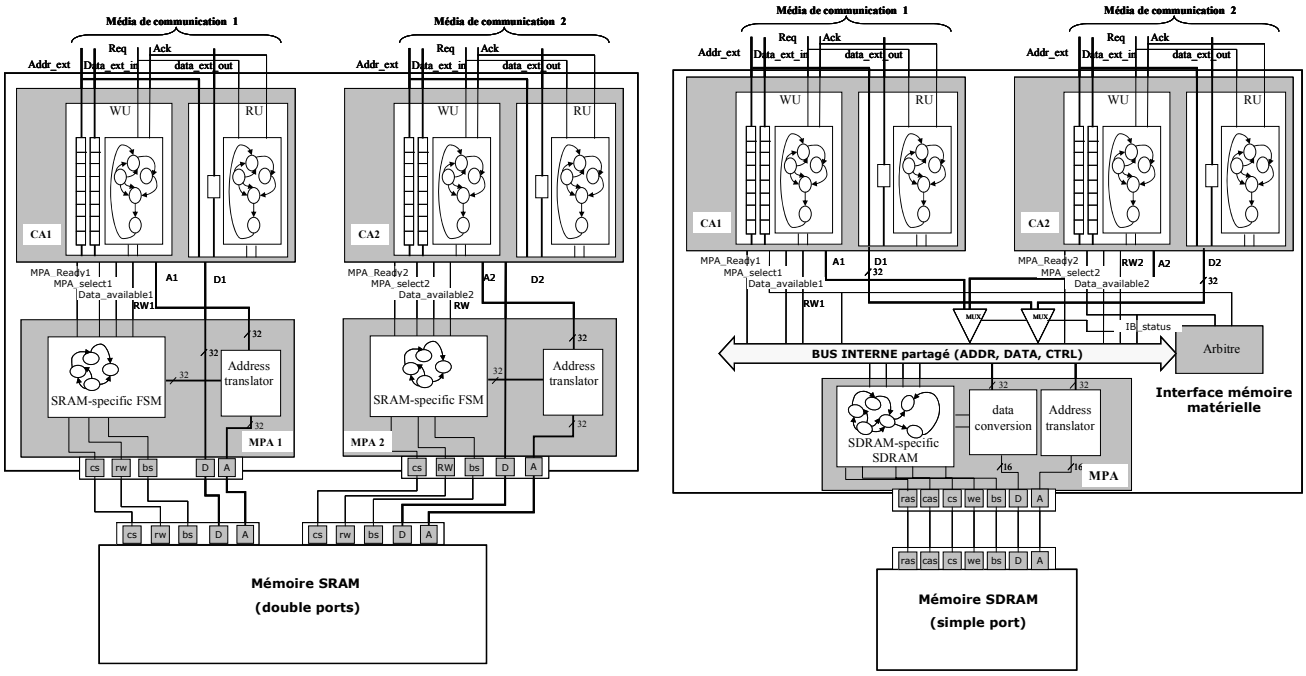

Figure 11 : Interfaces mémoire matérielles pour 2 architectures mémoire

V.4 Travaux sur les interfaces logicielles

V.4.1 Architecture générique

Les pilotes d'accès mémoire permettent au code de l'application d'accéder à la mémoire globale à travers les couches matérielles. L'architecture de ces pilotes logiciels est structurée en quatre couches superposées pour assurer leur flexibilité et limiter leur complexité. En effet, cette structuration permet la séparation explicite entre le code des pilotes dépendant et indépendant du processeur. Ceci facilite la réutilisation du code et par conséquent, améliore la productivité du logiciel.

Comme l'indique la Figure 12, les quatre couches d'un pilote logiciel correspondent aux parties suivantes : API du pilote, des services dépendants du logiciel de l'application, des éléments de mémorisation internes au pilote et une partie implémentant le logiciel dépendant du matériel.

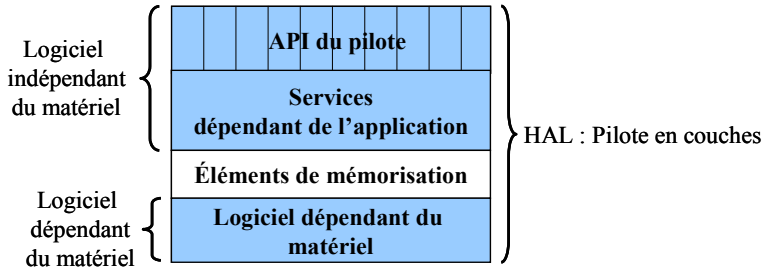

Figure 12 : Architecture en couches d'un pilote logiciel

L'API est la couche supérieure du pilote qui correspond à une interface de programmation standard entre le programmeur de l'application et le matériel. L'utilisateur code les tâches de son application en utilisant ces API sans se soucier des détails de l'architecture (mémoire, SE, CPU, …). Les API cachent les détails d'implémentation des protocoles de communication et de transfert de données. Elles définissent les services de communication de haut niveau permettant l'accès à la mémoire globale. Ces services correspondent aux méthodes de transfert de données et de contrôle spécifiques aux protocoles de communication utilisés par l'application.

La couche services dépendants de l'application implémente les services de contrôle et de synchronisation nécessaires à la réalisation des API. Ces services sont dépendants du logiciel de l'application car ils utilisent des fonctions fournies par le système d'exploitation ou par une autre entité logicielle. Lorsque les tâches s'exécutent en parallèle, le pilote nécessite des services de gestion d'accès parallèles et de gestion d'interruptions fournis par le système d'exploitation. Les services de cette couche implémentent des fonctions d'écriture et de lecture permettant d'accéder aux éléments de mémorisation internes au pilote.

Les éléments de mémorisation représente la dernière frontière entre le logiciel indépendant du matériel et celui qui en est dépendant. Le logiciel indépendant du matériel écrit les données dans les éléments de mémorisation où le logiciel dépendant du matériel vient lire ces données. Les éléments de mémorisation sont des FIFO, un tampon mémoire ou de simples registres. Leur taille est spécifique à l'application. Actuellement, nous utilisons de simples registres pour stocker les adresses et les données envoyées pendant une opération d'écriture ou de lecture dans une mémoire globale.

Cette partie implémente les fonctions d'écriture et de lecture spécifiques aux processeurs. Ces fonctions sont liées aux protocoles de transfert de données et de gestion d'interruptions supportés par le processeur. Lors d'une opération d'écriture dans la mémoire globale, cette couche récupère l'adresse et la donnée à partir des registres du pilote et elle les adapte2 avant de les écrire dans les ports matériels du processeur cible.

V.4.2 Exemple d'écriture et d'utilisation

Dans l'exemple de la Figure 13, deux tâches T1 et T2 accèdent à une mémoire globale en lecture et en écriture en utilisant des primitives d'accès de haut niveau. L'implémentation de ces primitives correspond au code d'un pilote d'accès à la mémoire. L'architecture matérielle est composée d'une mémoire globale SRAM, d'un processeur ARM7 TDMI et d'un réseau de communication point à point. L'implémentation de l'architecture logicielle correspond au code C

[2] L'adaptation consiste à convertir les adresses logiques en adresses physiques, et à découper la donnée selon la taille du bus de données du processeur.

des différentes couches du pilote. Deux API (Put_shm_external, Get_shm_external) sont déclarées dans une classe C++. La définition de la fonction est indépendante du matériel et utilise un service HAL_Register_Put pour écrire adresses et données dans les registres de la couche de mémorisation. L'écriture dans ces registres est gardée par des service de synchronisation (Block, UnBlock) fournis par le système d'exploitation. La fonction ARM7_Write est spécifique au processeur choisi, et accède au port du processeur.

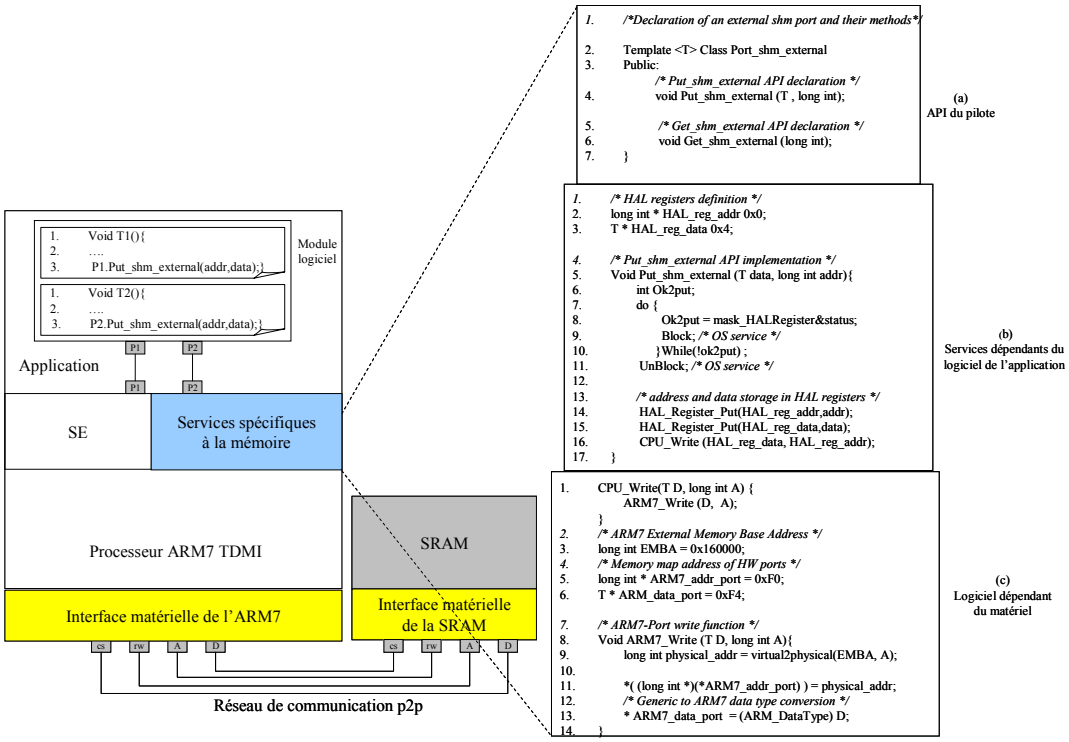

Figure 13 : Exemple de pilote en couches d'accès mémoire

V.4.3 Outils de génération des interfaces logicielles

Le flot de génération des pilotes logiciels (Figure 14) a été utilisé initialement pour la génération des systèmes d'exploitation [GAU01]. Nous l'avons étendu aux pilotes logiciels en couches pour la

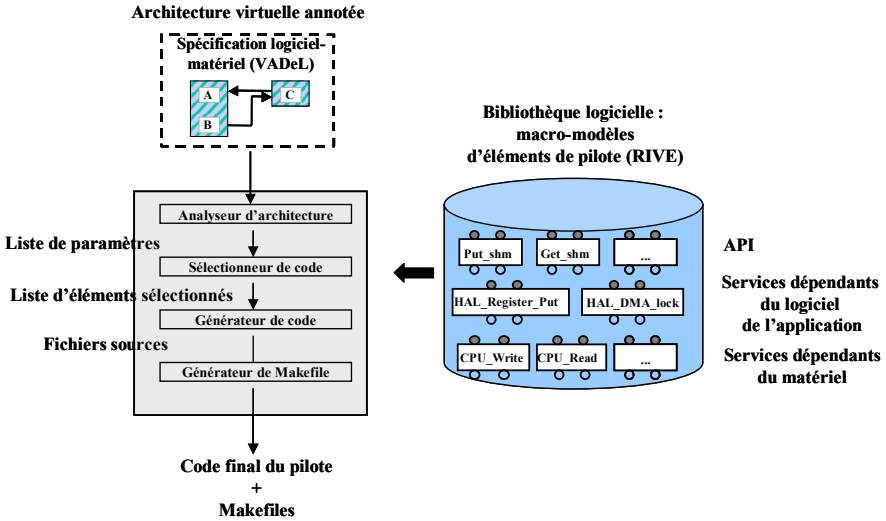

Figure 14 : Flot de génération des pilotes logiciels

mémoire. Comme dans le cas de génération des interfaces mémoire matérielles, le flot de génération des pilotes logiciels prend en entrée une architecture virtuelle annotée avec des paramètres de configuration spécifiques à l'application et à l'architecture (CPU, interfaces, …). Ces paramètres sont utilisés pour sélectionner et configurer les éléments spécifiques aux trois couches du pilote nécessaires au fonctionnement de l'application.

La génération des pilotes utilise une bibliothèque logicielle contenant des macro modèles d'éléments de chaque couche du pilote. Ces éléments génériques sont configurés par les paramètres d'annotation pendant la génération du code. Chaque élément peut fournir/requérir un service à/de un autre élément ce qui nécessite une relation de dépendance entre les services d'un pilote. Le code des macro modèles de cette bibliothèque est écrit en langage RIVE, le langage cible étant du C. La Figure 15 montre une partie du code macro de la fonction d'écriture sur un port spécifique à un processeur, avec le code macro de la bibliothèque (a), les paramètres de configuration utilisés (c), et le code C généré (b).

```
/* CPU-Port write function */
void @{CPUName}@_Write (T D, @{C_data_type}@ A){
      …….
      /* Address conversion */
       @{C_data_type}@ physical_addr = virtual2physical( @{EMBA}@, A);

        /* Address CPU port write */
       *( (@{C_data_type}@ *)(*CPU_addr_port) ) = physical_addr;

       /* Generic to CPU data type conversion and data CPU port write */
       * @{CPU_data_port}@  = (@{C_data_type}@) D;
      …….
}
```

(a) Code macro en RIVE

```
/* CPU-Port write function */
void ARM7_Write (T D, long int A){
      …….
      /* Address conversion */
       long int physical_addr = virtual2physical( 0x160000, A);

        /* Address CPU port write */
       *( (long int *)(*ARM7_addr_port) ) = physical_addr;

       /* Generic to CPU data type conversion and data CPU port write */
       * ARM7_data_port  = (long int) D;
      …….
}
```

(b) Code C généré

```
DEFINE CPUName = "ARM7" ENDDEFINE
DEFINE C_data_type = "long int" ENDDEFINE
DEFINE EMBA = 0x160000 ENDDEFINE
DEFINE CPU_addr_port = "ARM7_addr_port" ENDDEFINE
DEFINE CPU_data_port = "ARM7_data_port" ENDDEFINE
```

(c) Paramètres de configuration

Figure 15 : Exemple d'un macro-modèle du pilote écrit en RIVE

V.5 Avantages et limitations de ces travaux

Le filtre d'images développé par F. Gharsalli dans ses travaux de thèse [GHA03] autour de deux processeurs ARM7TDMI nous a permis de valider notre approche de génération des interfaces matérielles pour la mémoire globale nécessaire dans cette application. Plusieurs versions ont été générées selon le type de mémoire (SDRAM simple port, SRAM double port). La complexité des interfaces est de l'ordre de 2500 portes. La pénalité temporelle pour traverser les interfaces est de l'ordre de deux à trois cycles (horloge processeur ou horloge bus), ce qui reste faible vis-à-vis d'un accès classique en lecture (cinquantaine de cycles pour une SDRAM). Ce résultat est plus discutable pour une SRAM.

Le temps de génération ne prend que quelques minutes, mais ceci ne correspond qu'à la durée d'exécution de l'outil. Le temps de l'écriture manuelle de la spécification est de l'ordre de deux ou trois jours à condition de connaître le langage de spécification. Mais la tâche la plus longue est l'écriture de la bibliothèque de macro modèles. Les causes de cette difficulté sont :

- L'écriture d'un macro modèle nécessite un effort considérable pour couvrir toutes les configurations possibles,
- La validation globale au niveau cycle près d'une interface matérielle nécessite un temps important pour la vérification de la synchronisation entre les différents composants de l'architecture.

La bibliothèque de pilotes est composée d'éléments et de services. Chaque élément fournit des services et requiert des services fournis par d'autres éléments. La relation de dépendance entre ces éléments est représentée par un langage spécifique. Pour cette raison, l'ajout d'un nouvel élément à la bibliothèque nécessite la définition de ces relations avec les autres éléments de la bibliothèque déjà existants. La description doit aussi définir les services fournis et requis par l'élément ajouté, les paramètres d'appel de chaque méthode du pilote et les liens vers les sources d'implémentation. Ces modifications sont faites manuellement aujourd'hui, ce qui ne garantit pas l'exhaustivité des cas pris en compte.

La principale limitation de la génération des interfaces matérielles est liée à la bibliothèque des éléments de base (CA et MPA). Chacun de ces composants est spécifique à un comportement donné, et étroitement lié au réseau de communication ou au composant mémoire. Cette granularité assez grosse limite la réutilisation et impose l'écriture de nouveaux éléments à chaque évolution ou modification. De plus, et dans un contexte plus large que les interfaces matérielles pour la mémoire, les éléments de type MPA sont uniquement dédiés à un processeur ou à un composant mémoire.

V.6 Généralisation aux interfaces de communication matérielles

V.6.1 Nouveaux objectifs

La solution précédente de génération des interfaces matérielles a montré quelques limitations, notamment un manque de flexibilité du à des éléments de bibliothèque de trop grosse granularité et trop spécifiques. On cherche donc avec les travaux d'Arnaud Grasset (thèse en cours) à contourner ces limitations, en proposant des éléments de bibliothèque de granularité plus fine, ce qui permet de les réutiliser plus facilement. Bien entendu, la technique de conception change, ce qui suppose de revoir la spécification et la technique de composition.

Pour généraliser notre technique de génération des interfaces matérielles, il nous faut, dans un premier temps, trouver un modèle capable de modéliser des protocoles de communication. Ce modèle doit être facilement manipulable pour spécifier ce composant. Il doit être indépendant de la réalisation (logiciel ou matériel). Ensuite, toujours en utilisant la technique de composition d'éléments de bibliothèque, il faut utiliser un outil de composition pour produire un modèle RTL à partir des spécifications. Pour une plus grande flexibilité et une meilleure réutilisation, les éléments de bibliothèque seront plus petits, ce qui pourrait faciliter l'ajout de nouveaux composants.

L'approche par assemblage [LYO01][HOM01][VER96] utilise une spécification du système, à partir de laquelle on extrait un certain nombre de paramètres tels que les protocoles. Des composants de bibliothèques sont alors sélectionnés, configurés et assemblés pour générer l'interface de communication. L'efficacité de ces méthodes est limitée par le contenu des bibliothèques et par l'utilisation d'un seul modèle d'assemblage. Néanmoins, cette technique semble plus performante que l'approche par synthèse [OBE01][SEA96][PAS98][SIE02] qui utilise une description formelle de l'interface à partir de laquelle est extraite une machine d'états finis. Cette méthode offre un haut niveau d'abstraction pour la spécification mais leur conception reste manuelle ou restreint les interfaces générées.

V.6.2 De la spécification de service à la description synthétisable

La solution retenue est basée sur le modèle de description de protocoles de [ZIT93], qui permet de représenter des protocoles pour les réseaux de communication. L'élément de base est une fonction de protocole (FP) qui correspond à un composant fonctionnel (Figure 16). Chaque fonction de protocole fournit ou requiert des services de/vers d'autres fonctions de protocole et possède des paramètres de configuration. Les services d'une fonction de protocole sont accessibles par un port d'accès de service (SAP), et chaque port possède une liste de services rendus ou requis. Un protocole est alors modélisé par la composition de plusieurs fonctions de protocole.

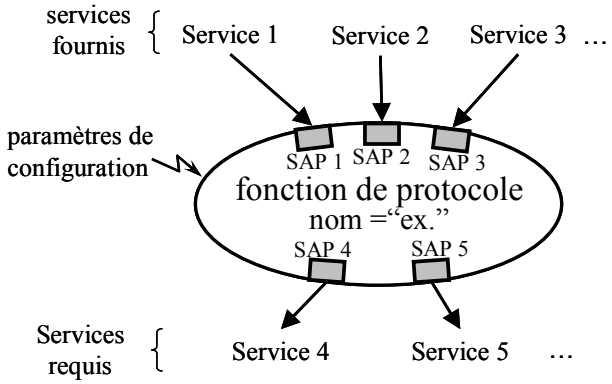

Figure 16 : Modèle de fonction de protocole

Un protocole de communication est décrit par un ensemble de fonctions de protocole reliées selon les services requis ou fournis. C'est encore une étape manuelle que de choisir les fonctions de protocole les plus appropriées.

Une des difficultés ensuite est de passer rapidement de cette représentation sous forme d'un ensemble de fonctions de protocole à une description RTL. Rappelons que ce modèle n'a à notre connaissance jamais été utilisé pour concevoir du matériel. La différence avec la conception de logiciel vient principalement de l'architecture du circuit, à déterminer, et de l'échange de données entre les différents éléments de cette architecture matérielle.

La solution proposée dans la thèse d'Arnaud Grasset repose sur un flot de conception systématique en plusieurs étapes. Il ne s'agit pas de définir un outil ou un flot de synthèse, au sens synthèse d'architecture ou synthèse logique, mais bien une méthode d'assemblage de composants existants et paramétrables. Pour cela, au moins un modèle RTL est associé à chaque fonction de protocole. Le flot peut donc se résumer au quatre étapes suivantes (Figure 17) :

- La sélection de composants consiste à choisir le modèle RTL associé à chaque fonction de protocole,
- Les valeurs de paramètres de configuration sont calculées dans l'étape de configuration (taille des données, des bus, …),
- Le principe de la génération de connexions est de remplacer les fonctions de protocole par les composants RTL paramétrés. La difficulté est alors de générer les interconnexions entre ces composants, ainsi que les signaux d'horloge et de reset.
- On obtient enfin le modèle HDL pendant la génération de code (expansion de macro code).

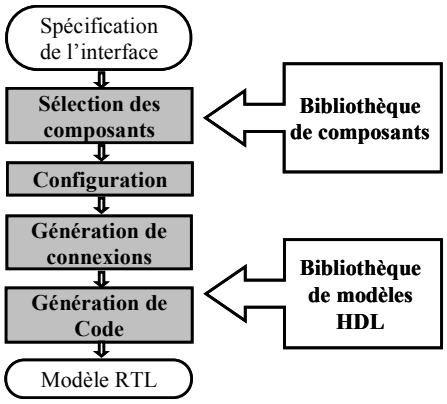

Figure 17 : Flot de génération des interfaces matérielles

V.6.3 Avantages et limitations

La génération automatique du modèle RTL à partir de la spécification facilite l'exploration de l'architecture du système. En effet, ce flot rapide autorise le concepteur à essayer plusieurs architectures systèmes ou protocoles, sans que la conception des interfaces matérielles soit un frein. La technique d'assemblage proposée est relativement simple, puisqu'il ne s'agit que d'assembler des composants existants, l'architecture de l'interface dépendant fortement du modèle de protocole. La partie "intelligente" reste donc à la charge du concepteur.

Comme toujours dans les techniques d'assemblage de composants, une des limitations provient du contenu des bibliothèques. Il faut enrichir ces bibliothèques selon les besoins, et si la génération elle-même ne prend que quelques secondes ou millisecondes, l'écriture des éléments de bibliothèque est la tâche la plus difficile. Néanmoins, les éléments de bibliothèque sont plus petits que ceux de la version initiale d'ASAG. On peut donc attendre qu'ils soient plus faciles à écrire et à valider, moins spécifiques, et donc plus facilement réutilisables.

Cette technique fait apparaître la possibilité de modéliser (ou spécifier) un protocole de communication sans se soucier de la frontière entre logiciel et matériel, c'est-à-dire entre pilote logiciel et interface matérielle. Néanmoins cette frontière existe, et elle amène une question : Comment déterminer la frontière logiciel/matériel dans une spécification de protocoles, ce qui nous ramène à un problème de partitionnement logiciel/matériel avec les résultats décrits dans le chapitre III.

V.7 Bilan, conclusion et perspectives

L'objectif de ces travaux sur la génération des interfaces matérielles (et logicielles) est de faciliter la conception de ces composants d'interface, pour que celle-ci ne soit plus un frein à la phase d'exploration d'architectures. Cet objectif est atteint en ce qui concerne les interfaces matérielles et les expérimentations ont montré l'intérêt de ces travaux. Des travaux supplémentaires sont nécessaires pour les interfaces logicielles.

Une nouvelle problématique est apparue qui pourrait être un des prochains thèmes de recherche sur les interfaces logiciel/matériel. Le fait de modéliser un protocole indépendamment de sa réalisation future suppose de décider de la limite entre parties matérielles et logicielles. Une telle décision a des conséquences sur la partie logicielle de bas niveau et sur l'architecture de l'interface matérielle. Notre bonne connaissance des interfaces logiciel/matériel devrait maintenant nous permettre d'aborder efficacement cette nouvelle problématique.

V.8 Principales publications relatives au chapitre V

[RN1] F. Gharsalli, F. Rousseau, A. Jerraya, *Conception des interfaces logiciel-matériel pour l'intégration des mémoires globales dans les systèmes monopuces*, Techniques et Sciences Informatiques, à paraître en 2005.

[CI2] A. Grasset, F. Rousseau, A. Jerraya, *Automatic Generation of Component Wrappers by Composition of Hardware Library Elements Starting from Communication Service Specification*, Workshop on Rapid System Prototyping (RSP 2005), Montréal, Canada, Juin 2005.

[CI3] A. Grasset, F. Rousseau, A. Jerraya, *Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation*, Workshop on Rapid System Prototyping (RSP 2004), pp. 66-69, Genève, Suisse, Juin 2004.

[CI5] F. Gharsalli, S. Meftali, F. Rousseau, A. Jerraya, *Unifying Memory and Processor Wrapper Architecture in Multiprocessor SoC Design*, International Symposium on System Synthesis (ISSS 2002), pp. 26-31, Kyoto, Japon, Oct. 2002.

[CI6] F. Gharsalli, S. Meftali, F. Rousseau, A. A. Jerraya, *Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC*, Design Automation Conference (DAC 2002), pp. 596-601, La nouvelle Orléans, EU, Juin 2002.

V.9 Encadrement de thèse

Ferid GHARSALLI. *Conception des interfaces logiciel/matériel pour l'intégration des mémoires globales dans les systèmes monopuces*. Doctorat de l'INPG, soutenue le 1er juillet 2003. En co-direction avec Ahmed Jerraya. Taux d'encadrement de 90%.

Arnaud GRASSET. *Synthèse des interfaces matérielles dans la conception des systèmes monopuces : de la spécification à la génération automatique*. Doctorat de l'INPG, thèse en cours, soutenance prévue en novembre 2005. En co-direction avec Ahmed Jerraya. Taux d'encadrement de 90%.

VI Prototypage sur une plateforme reconfigurable

Les limites de la simulation nous ont amené à réfléchir sur une réalisation des systèmes pour valider nos approches de conception. Le prototypage sur une plateforme reconfigurable a été initié avec l'acquisition d'une plateforme multiprocesseur (ARM Integrator) développée pour valider le logiciel d'applications embarquées. Cette plateforme a été prise en main par Arif Sasongko (fin 2001) pour mener son travail de thèse. Des travaux sont toujours en cours avec Benaoumeur Senouci.

VI.1 Complexité de la vérification d'un système multiprocesseur monopuce

La vérification d'un système multiprocesseur monopuce est un processus exigeant à cause de sa complexité et de son hétérogénéité. Il faut des techniques qui permettent de vérifier plusieurs aspects : fonctionnel et architectural, logiciel et matériel, ainsi que la communication. La vérification a lieu tout au long de la conception :

- Vérification de la fonctionnalité au niveau des spécifications comportementales,
- Vérification de l'architecture du système pendant l'étape de partitionnement logiciel/matériel,
- Vérification des fonctionnalités des composants créés pendant les phases de synthèse,
- Vérification de l'intégration des composants en associant tous les composants,
- Vérification du fonctionnement du système dans son environnement.

Le coût global de vérification de tels systèmes est très élevé en terme d'effort et de temps. En effet, on estime que 70% du temps de conception est dédié aux diverses vérifications.

Les systèmes monopuces contiennent à la fois des composants matériels complexes (processeurs, décodeurs vidéo, audio, périphériques, …) et une énorme quantité de logiciel. On considère que 70% des concepteurs passent 50% de leur temps pour développer la partie logicielle. Ce logiciel est tellement complexe que sa vérification n'est plus réaliste en utilisant des approches classiques de simulation basée sur des ISS (simulateur de processeurs).

VI.2 Intérêt et objectifs du prototypage sur une plateforme reconfigurable

La solution la plus naturelle pour vérifier le logiciel sur l'architecture matérielle sur laquelle il est exécuté est d'attendre de posséder cette architecture. Si le développement du matériel et du logiciel peut alors être fait en parallèle, l'intégration et la vérification du logiciel est faite après la fabrication de la puce. Cette séquentialité est très pénalisante pour une mise sur le marché rapide d'un nouveau produit.

Pour accélérer la mise sur le marché, il est important de commencer la vérification du logiciel plus tôt, en tout cas avant que la puce soit fabriquée. Pour ce faire, une solution est de développer en parallèle avec la puce, une architecture, dite de prototypage qui permet de vérifier certaines fonctionnalités du logiciel. Le prototypage consiste à réaliser le système sur une architecture différente de celle prévue pour la réalisation finale. Tous les composants ou sous-systèmes du système final sont remplacés soit par des composants physiques (logiciels ou matériels), soit sont émulés ou simulés. Mais le développement d'un prototype spécifique à une application est lui aussi très coûteux.

On préfère alors utiliser une plateforme de prototypage générique permettant de réaliser plusieurs applications. Une plateforme reconfigurable nécessite un petit effort de configuration pour correspondre à l'architecture finale, mais elle a le mérite d'exister physiquement. Et le temps de configuration est très faible comparé au temps de conception. L'utilisation d'une plateforme de prototypage reconfigurable semble être une bonne solution pour vérifier plus tôt dans le cycle de conception une partie du logiciel (Figure 18).

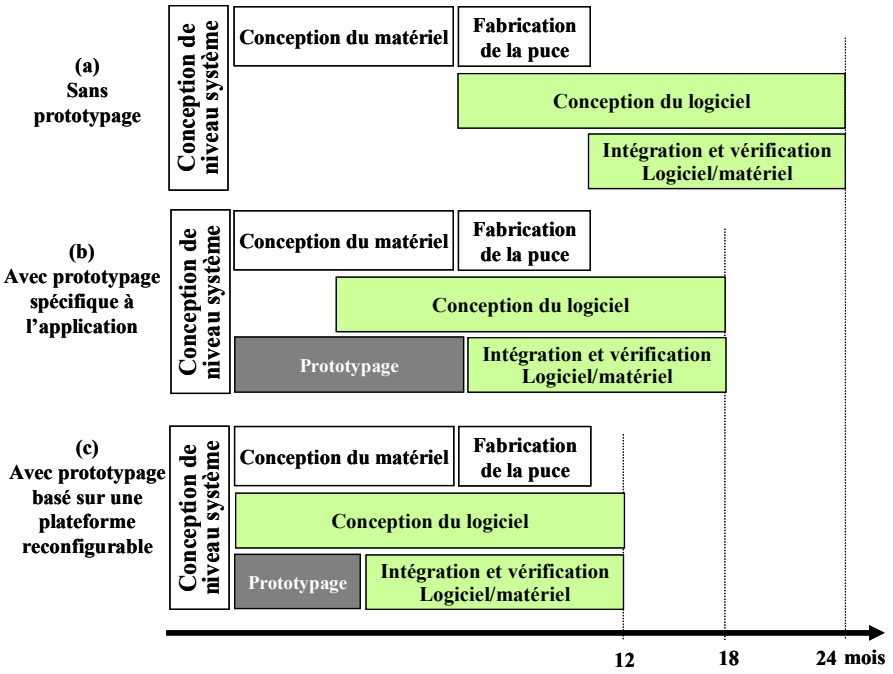

Figure 18 : Effet du prototypage sur la conception d'un système monopuce

Les travaux de thèse d'Arif Sasongko [SAS04], poursuivis par ceux de Benaoumeur Senouci s'attachent à définir un flot de prototypage sur une plateforme reconfigurable ainsi qu'une automatisation de ce flot pour permettre de passer rapidement d'une spécification du système à un prototype. L'intérêt de proposer un flot de prototypage est de définir l'enchaînement des différentes étapes pour conduire au prototype d'une façon si possible systématique. L'automatisation de certaines de ces étapes a pour objectif d'accélérer la conception, et de faciliter le travail du concepteur en lui évitant des tâches difficiles et répétitives.

L'objectif principal du prototypage reste, de notre point de vue, la validation du logiciel. Il s'agit par exemple de valider le code d'une application parallèle, ou le système d'exploitation, ou un mécanisme de synchronisation et de communication. Cette vérification peut s'appliquer aussi aux couches les plus basses du logiciel à condition que l'architecture de la plateforme soit très proche de l'architecture du système à valider (même processeur, même organisation mémoire, …), et dans ce cas une validation des interfaces logiciel/matériel est possible.

VI.3 Modèles et flots de prototypage

VI.3.1 Principes généraux

Le prototypage est une réalisation préliminaire d'un système sur une cible différente de la cible de réalisation finale [HAR97a][CHA99]. Tous les composants de l'architecture finale doivent être inclus dans le prototype. La réalisation de ces composants peut être diverse : vrais composants logiciel/matériel, composants émulés, ou même composants simulés.

Le prototypage d'une application sur une plateforme reconfigurable consiste à réaliser toutes les parties de l'application avec les composants disponibles sur la plateforme. Les composants matériels de l'application (processeurs, circuits spécifiques) sont réalisés avec des composants programmables (FPGA). Les composants logiciels sont réalisés par des programmes exécutés par un ou plusieurs processeurs.

Selon la nature de l'application et l'architecture de la plateforme, ce processus est plus ou moins complexe. Pour réaliser un prototype à partir du modèle RTL de l'application, il est nécessaire d'avoir une méthode pour enchaîner les étapes des transformations du modèle RTL et de la plateforme. C'est le flot de prototypage.

VI.3.2 Modèle de systèmes multiprocesseurs monopuces

La Figure 19 a) montre le modèle d'architecture générique d'un système monopuce que nous utilisons pour le prototypage. Le développement de ce modèle est basé sur le fait que ces systèmes sont représentés par un ensemble de tâches. La conception conduit au niveau RTL dans lequel les tâches sont exécutées par des processeurs ou effectuées par des composants matériels spécifiques (IP). Aussi, le modèle d'un système monopuce comprend deux types de nœuds de calcul : Le nœud de calcul logiciel contient un processeur pour exécuter le logiciel et le nœud de calcul matériel contient un composant spécifique. Les données sont échangées entre les nœuds de calcul logiciel et matériel à travers un (ou plusieurs) réseau de communication, qui peut être un bus partagé, des connexions point à point, des "crossbars", un réseau sur puce ou une combinaison de ces éléments.

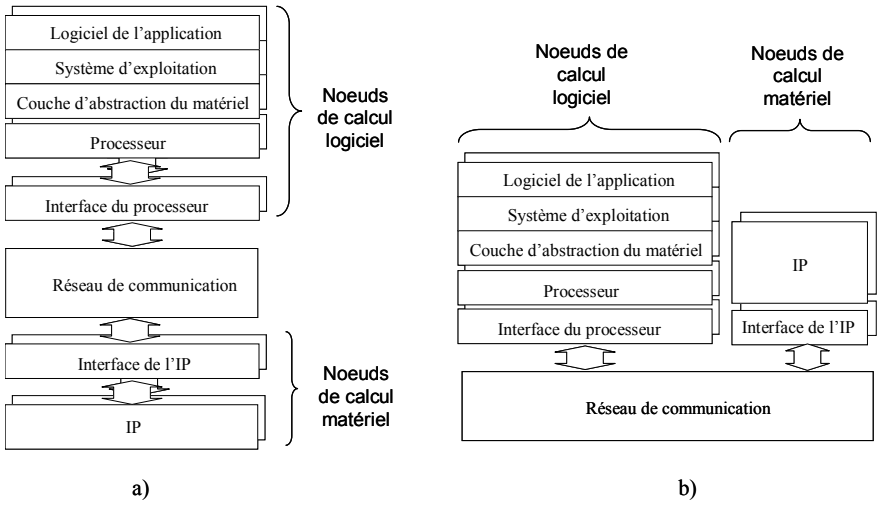

Figure 19 : modèle générique d'un système multiprocesseur monopuce

Un nœud de calcul logiciel contient le logiciel de l'application, un système d'exploitation (SE), une couche d'abstraction du matériel, le processeur, et l'interface du processeur. Le logiciel de l'application, la couche d'abstraction du matériel, et le SE sont les parties logicielles de l'application exécutés par le processeur. Le logiciel d'application est un ensemble de tâches utilisé pour construire le comportement de l'application décrit dans la spécification fonctionnelle. Le SE et la couche d'abstraction du matériel sont les parties logicielles qui gèrent l'exécution de ces tâches et l'utilisation de ressources partagées par les tâches. Ceci inclut l'utilisation du matériel. La couche d'abstraction du matériel est séparée du SE pour permettre de porter facilement le SE et le code de l'application sur différentes architectures. Quelques parties de la couche d'abstraction du matériel sont écrites en assembleur car très liées au processeur. Le reste est écrit en langage de programmation de haut niveau (C, C++, …). L'interface du processeur est là pour adapter l'interface du processeur au réseau de communication.

Le nœud de calcul matériel consiste en un composant IP qui effectue un calcul ou une tâche spécifique et son interface. L'interface de l'IP adapte celui-ci au réseau de communication. Ces nœuds de calculs sont décrits au niveau RTL ou au niveau portes logiques pour permettre le prototypage. La Figure 19 b) donne une autre représentation de ce modèle aussi utilisé dans ce mémoire.

VI.3.3 Modèle générique d'une plateforme de prototypage

La plateforme de prototypage comporte des nœuds pour exécuter les programmes de la partie logicielle, et des nœuds pour effectuer les tâches matérielles. La Figure 20 a) décrit un modèle générique de plateforme pour le prototypage. Dans ce modèle, on divise la plateforme de prototypage en trois parties : les nœuds de prototypage logiciel, les nœuds de prototypage matériel, et le réseau de prototypage de communication.

Le nœud de prototypage logiciel contient au minimum un processeur et de la mémoire. Le logiciel est ainsi chargé en mémoire et exécuté par le processeur. Ce noeud contient aussi l'interface pour se connecter au réseau de prototypage de communication.

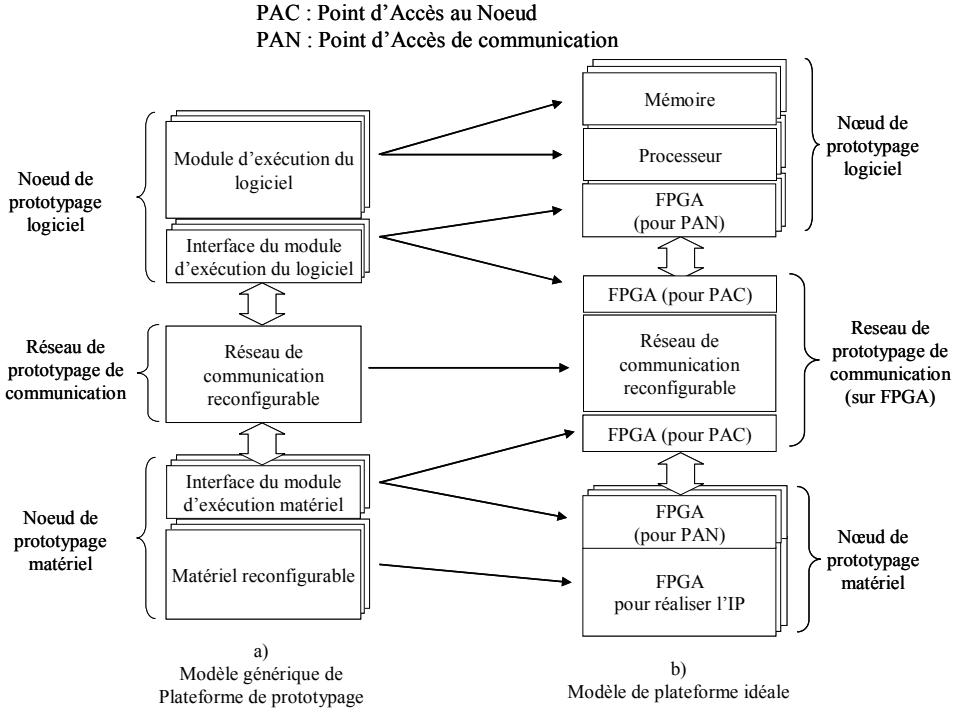

Figure 20 : Modèle générique de plateforme de prototypage et sa réalisation

Le nœud de prototypage matériel contient au moins un composant reconfigurable. La fonctionnalité d'un composant spécifique de l'application est reproduite par ce composant sous certaines conditions (description niveau RTL, …). Ce nœud a besoin aussi d'une interface pour se connecter au réseau de prototypage de communication.

Les nœuds de prototypage logiciel et matériel sont connectés à travers un réseau de prototypage de communication, qui permet de reproduire un ou plusieurs schémas de communication.

Pour être capable de modéliser des architectures variées, une plateforme reconfigurable doit avoir les caractéristiques suivantes :

- Etre modulaire, ce qui signifie que les types et le nombre de nœuds de la plateforme sont variables,

- Le nœud de prototypage logiciel peut remplacer tous les types de processeur et son architecture locale pour modéliser un nœud de calcul logiciel,
- Le nœud de prototypage matériel est capable de modéliser un IP et son interface,
- Le réseau de communication entre les nœuds peut être configuré pour réaliser divers types de communication.

La plateforme idéale est détaillée sur la Figure 20 b). Cette représentation fait apparaître les nœuds de prototypage comportant de la mémoire et un processeur, les nœuds de prototypage matériel basé sur un composant programmable (FPGA) et un réseau de prototypage de communication. Les points d'accès aux nœuds (PAN) et au réseau de communication (PAC) sont aussi réalisés avec des composants programmables.

Une plateforme de prototypage idéale doit permettre de prototyper n'importe quelle application. Ceci suppose qu'elle soit entièrement configurable, c'est-à-dire avec un nombre et des types de nœuds de prototypage matériel et logiciel variés, qui acceptent des configurations différentes. Le réseau de prototypage de communication doit aussi être configurable pour supporter un grand nombre de média de communication et de protocoles.

VI.3.4 Flot de prototypage simple

Quand l'architecture de la plateforme est très proche de l'architecture de l'application, le flot de prototypage est très simple et se limite à une étape d'allocation ("assignment") suivie d'une étape de génération de code, compilation et synthèse ("targeting") (Figure 21).

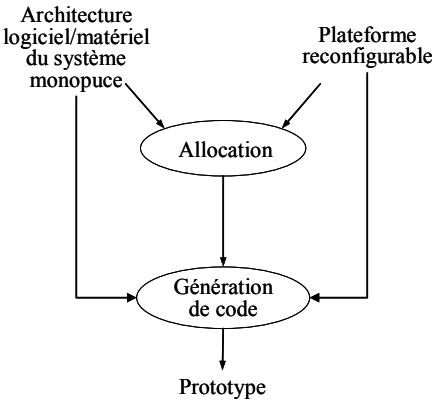

Figure 21 : Flot de prototypage simple

Dans l'étape d'allocation, les concepteurs associent chaque partie de l'architecture à un nœud de prototypage de la plateforme. Ces associations indiquent sur quelles parties de la plateforme sont réalisées les parties de l'architecture de l'application. Le résultat de cette étape est un tableau d'association qui guide toute la suite du processus de prototypage. Les décisions prises pour faire ces associations sont basées sur les objectifs du prototypage et les contraintes imposées par la plateforme. Ce n'est pas du partitionnement logiciel/matériel car l'application est déjà décrite à un bas niveau d'abstraction. Cette étape est faite manuellement car elle nécessite des prises de décisions et de la créativité, qu'il nous semble raisonnable de laisser au concepteur.

La génération de code est la dernière étape dans le flot de prototypage. Cette étape consiste en processus standard tels que la compilation et l'édition de lien, la synthèse logique, le placement sur FPGA, et le routage. De nombreux outils industriels sont disponibles et adaptés aux plateformes.

Ce flot simple est le seul proposé avec les plateformes commerciales de chez ARM, APTIX ou EVE. Il se limite aux étapes classiques de compilation, synthèse logique, partitionnement, placement et routage et programmation du mécanisme de communication.

VI.3.5 Flot de prototypage réel

Malheureusement, le flot de prototypage simple présenté précédemment ne s'applique que très rarement dans la réalité. De nombreuses différences existent entre l'architecture de la plateforme et celle de l'application : le réseau de communication de la plateforme fixé n'est pas celui de l'application, les processeurs ne sont pas du même type, la carte mémoire est différente et dans certains cas les programmes de démarrage, le traitement des exceptions et la gestion des interruptions sont à reconsidérer. Le prototypage nécessite alors de réorganiser la plateforme et éventuellement d'apporter quelques modifications à l'architecture de l'application.

Ainsi nous proposons un flot avec quatre étapes (Figure 22) : allouer chaque partie de l'application à la plateforme, configurer la plateforme pour qu'elle convienne à l'architecture de l'application, adapter l'architecture de l'application si nécessaire, et générer le code pour les nœuds de prototypage de la plateforme.

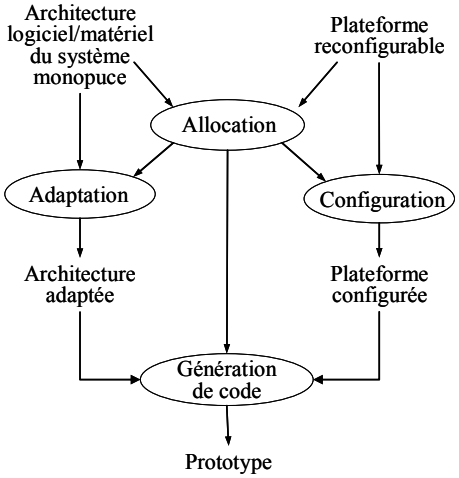

Figure 22 : Flot de prototypage réel

On définit la configuration comme une réorganisation de la plateforme pour que son architecture soit proche de celle de l'application. Cette réorganisation n'est pas toujours simple, par exemple quand le réseau de prototypage de communication est fixé et ne correspond pas à celui de l'application. Il faut dans ce cas ajouter des convertisseurs de protocoles, dans les différents composants programmables de la plateforme pour "cacher" ce réseau imposé.

L'adaptation consiste à modifier l'application pour satisfaire aux caractéristiques de la plateforme. Cette étape n'est mise en œuvre que si la configuration de la plateforme ne permet pas de s'adapter aux besoins de l'application. L'exemple classique est lié au plan mémoire fixé sur la plateforme et qui ne correspond pas à celui décidé pour l'application. Il faut alors modifier le plan mémoire dans l'application pour respecter les contraintes de la plateforme.

L'allocation et la génération de code sont des étapes identiques à celle présentées dans le flot de prototypage simple.

VI.3.6 Discussion sur l'adaptation et la configuration

La principale difficulté qui apparaît dans le prototypage sur une plateforme reconfigurable provient des contraintes imposées par la plateforme et des disparités entre l'architecture de l'application et celle de la plateforme. En effet, les plateformes de prototypage commerciales ou universitaires ne sont pas idéales et des contraintes (matériel et logiciel) existent qui limitent les possibilités de configuration. Les différences entre l'architecture de l'application et celle de la plateforme sont contournées en configurant la plateforme ou en adaptant l'application.

D'un point de vue plus technique, on peut citer quelques contraintes fortes des plateformes de prototypage : Plan mémoire fixé, nombre et priorité des interruptions fixés, contrôleur d'interruptions imposé, réseau et protocole de communication fixé, limitation sur le nombre et le type de processeurs, ….

La Figure 23 montre un exemple de configuration. Dans cet exemple, on veut prototyper une architecture (Figure 23.a) qui contient un processeur, un IP, et un bus X. La plateforme disponible consiste en un nœud de prototypage logiciel contenant le même type de processeur, et un nœud de prototypage matériel contenant un FPGA qui permet de réaliser l'IP. Malheureusement, la communication entre le processeur et le FPGA est différente (bus Y) et fixée. On doit alors implémenter le bus X dans le FPGA. Afin de le faire, on a besoin d'un convertisseur pour cacher le bus Y.

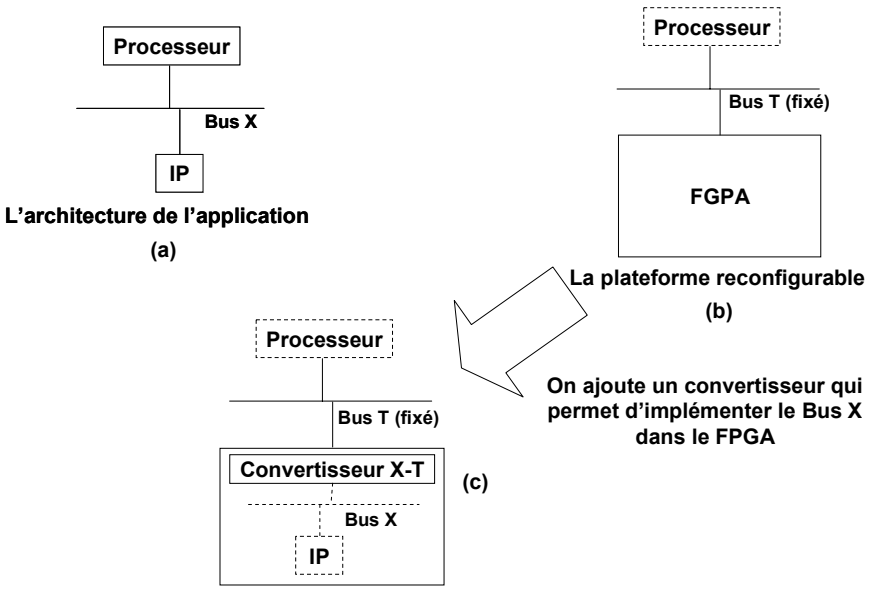

Figure 23 : Exemple de configuration

Si l'objectif du prototypage suppose de réaliser le bus X, alors il apparaît évident de placer le bus X ainsi que tous le composants d'interface de l'application dans le nœud de prototypage matériel. Il est donc nécessaire d'ajouter une couche de conversion entre le réseau Y et le bus X. Le réseau fixé devient transparent pour l'IP grâce à ce convertisseur en terme de protocole entre l'IP vers le processeur. Ceci n'est pas vrai pour les communications du processeur vers l'IP, puisque le nœud de prototypage logiciel est connecté au réseau Y.

Dans ce cas, la configuration consiste à développer le convertisseur entre le réseau fixé de la plateforme et l'interface du processeur. Ce convertisseur est réutilisable sur cette même plateforme.

La Figure 24 montre un exemple d'adaptation nécessaire dans le cas ou les adresses d'accès aux composants matériels sont réparties dans l'espace d'adressage pour l'application. Si ces composants matériels sont tous réalisés dans le FPGA d'un nœud de prototypage matériel, l'espace d'adressage de FPGA est fixé sur la plateforme. Il y a donc lieu de modifier l'application pour s'adapter à la plateforme.

Si cet exemple d'adaptation peut sembler trivial, d'autres cas sont plus difficile à traiter. Par exemple, les modifications de code de "boot", les pilotes de périphériques, les fonctions de lecture/écriture à travers le réseau, les routines de traitement des interruptions, le nombre réduit d'interruptions disponibles demandent tous des modifications difficiles à effectuer. Ces difficultés sont liées à plusieurs facteurs. Elles supposent toutes une très bonne connaissance de la plateforme, de l'architecture et de la description bas niveau de l'application. On remarque aussi

que tous les exemples d'adaptation concernent la modification des couches basses logicielles, généralement écrites en assembleur et très proche des processeurs. Pour ces raisons, la configuration est préférée à l'adaptation pour résoudre une incompatibilité, ce qui évite alors de modifier le code de l'application.

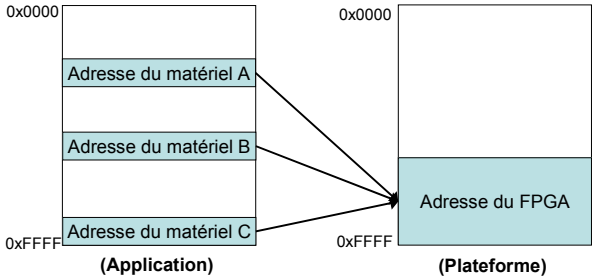

Figure 24 : Exemple d'adaptation

VI.4 Difficultés de l'automatisation de l'adaptation

L'adaptation de l'application consiste donc à modifier certaines couches logicielles de bas niveau, ce qui peut entraîner des erreurs, prend du temps et donc ralentit le processus de prototypage. Les expériences sur les applications VDSL et DivX montrent par exemple qu'environ 20% du code du programme de boot est à reprendre (adressage mémoire, interruptions, initialisation des piles). De même, dans ces deux applications, les modifications affectent une centaine de fichiers, ce qui n'est pas raisonnable à faire manuellement.

Une solution pour simplifier et accélérer cette étape est de l'automatiser, c'est-à-dire d'écrire des outils permettant de faire automatiquement ces transformations. Certaines transformations sont triviales et ne présentent pas de difficulté (translation d'adresses), mais ce n'est pas le cas dans la mise en place de protocoles de communication complexes (Message Passing Interface par exemple) pour lesquels on peut être amener à remplacer des fonctionnalités matérielles par du logiciel.

Mais en se rappelant que l'outil ROSES est capable de générer les couches logicielles de bas niveau pour l'application, l'idée qui prévaut dans la thèse d'Arif Sasongko [SAS04] est d'utiliser ROSES pour générer les couches logicielles de bas niveau pour la plateforme.

Cette technique consiste à reprendre le flot de conception, et à modifier les spécifications de l'application en donnant les paramètres correspondant à la plateforme (Figure 6). Le fonctionnement de ROSES repose sur un assemblage d'éléments de bibliothèque, et suppose donc que tous les éléments de base sont contenus dans la bibliothèque.

Mais pour utiliser cette technique, la bibliothèque doit avoir les éléments pour réaliser tous les services requis sur la plateforme. Ces éléments doivent avoir les mêmes comportements que sur la puce finale. L'utilisation d'élément de bibliothèque est décrite dans [GAU01]. A partir d'un service requis par une tâche, l'outil ASOG détermine l'élément utilisé. Il est possible d'avoir plusieurs réalisations de cet élément. Chaque élément doit posséder dans la bibliothèque au moins deux solutions de réalisation, l'une pour le système monopuce, l'autre pour la plateforme. En pratique, il y en a plus car un élément ou un service peut être découpé en parties logicielle et matérielle avec des frontières différentes [PAV04]. Il faut aussi s'assurer de la validation de ces éléments de bibliothèque et leur compatibilité, ce qui reste une difficulté non abordée dans la thèse d'Arif Sasongko.

VI.5 Bilan, conclusion et perspectives

L'intérêt d'une adaptation automatique est d'obtenir au plus vite le prototype pour développer et valider ou optimiser au plus tôt les parties logicielles. L'exécution du logiciel d'application sur ce prototype présente des avantages par rapport à la co-simulation. La co-simulation en utilisant un ISS et un simulateur (VHDL par exemple) est très lent. Il est alors difficile de vérifier le logiciel d'application, souvent très complexe et dont l'exécution peut prendre beaucoup de temps. La plateforme permet d'observer les effets du parallélisme et de valider la synchronisation entre les tâches. On vérifie aussi les parties du système d'exploitation qui sont indépendantes du matériel : Ordonnanceur, les sémaphores, la pile, …… Mais, ce qui dépend du matériel est difficile, voire même impossible à vérifier : la latence d'une d'interruption, compteur de temps, …. .

Le prototypage tel qu'il est décrit dans ce mémoire ne permet pas bien sur le développement ou la validation de la couche logicielle d'abstraction du matériel, puisque celle de la plateforme est différente de celle de l'application finale.

On constate que l'adaptation est difficile à faire, mais cette adaptation est nécessaire quand aucune configuration de la plateforme n'aboutit à une architecture satisfaisante. Il faut donc se diriger vers une plateforme encore plus reconfigurable, et tendre vers une plateforme idéale.

VI.6 Principales publications relatives au chapitre VI

[RI1] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *Towards SoC Validation Through Prototyping: A Systematic Approach Based on Reconfigurable Platform*, Design Automation for Embedded Systems, vol. 8, pp 155-171, Kluwer Academic Publishers, déc. 2003.

[R1] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *SoC Validation trough Prototyping on ARM Integrator Platform*, ARM Information Quarterly (IQ), volume 2, numéro 2, 2003.

[CI4] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *Embedded Application Prototyping on a Communication-Restricted Reconfigurable Platform*, Workshop on Rapid System Prototyping (RSP 2003), pp. 33-39, San Diego, EU, juin 2003.

VI.7 Encadrement de thèse

Arif SASONGKO. *Prototypage basé sur une plateforme reconfigurable pour la vérification des systèmes monopuces*. Doctorat de l'Université Joseph Fourier– Grenoble, soutenue le 15 octobre 2004. En co-direction avec Ahmed Jerraya. Taux d'encadrement de 90%.

Benaoumeur SENOUCI. *Automatisation du flot de prototypage basé sur une plateforme reconfigurable pour la validation des interfaces logiciel/matériel*. Doctorat de l'INPG, thèse en cours, soutenance prévue en octobre 2007. En co-direction avec Frédéric Pétrot.

VII Conclusion

VII.1 Bilan

Ce mémoire fait un bilan d'une dizaine d'années de recherche. Si le cadre de mes travaux a toujours été la conception des systèmes logiciel/matériel, quatre grands axes ont été explorés : le partitionnement logiciel/matériel, les architectures mémoire, les interfaces matériels de communication et le prototypage.

Le partitionnement logiciel/matériel est devenu une activité marginale, avec encore quelques travaux. De nombreuses méthodes et algorithmes ont vu le jour à la fin des années 90 sans que ces travaux n'apportent de solutions satisfaisantes. L'évolution des techniques de conception, de synthèse et de compilation, les caractéristiques des composants et la complexité des applications et des architectures sont certainement responsables de l'inefficacité des solutions dans ce domaine.

Les premières applications traitées pendant mon travail de thèse avaient déjà fait apparaître la nécessité de bien prendre en compte l'architecture mémoire dans la conception de systèmes pour des raisons de performances et de surface. C'est encore plus vrai maintenant avec les applications multimédias traitant des flux de données importants. Les travaux menés dans l'équipe SLS nous ont permis de bien comprendre l'importance de l'architecture mémoire et son influence sur les performances. Une solution est apportée pour rechercher l'architecture mémoire optimale pour une application donnée. Elle est discutable aujourd'hui dans ses hypothèses de départ, un peu trop restrictives quant aux applications et aux composants visés. Un travail complémentaire est nécessaire pour faire évoluer la méthode qui semble néanmoins s'appliquer dans un spectre de conception plus large.

Les travaux sur la conception (ou la génération) des interfaces matérielles sont une continuité avec les premières études sur les interfaces matérielles des processeurs. L'évolution vers les composants mémoire nous a permis d'asseoir notre connaissance du problème et a mis en évidence quelques limitations que nous essayons de gommer avec la généralisation à tous les composants. La méthode originale proposée -reposant sur la spécification sous forme de services requis et fournis- pèche encore par certains aspects pour être efficacement utilisée. Une estimation de performance est nécessaire pour rapidement évaluer la qualité de la solution et des travaux sont en cours. De plus, l'intégration dans le flot ROSES se poursuit pour obtenir un outil automatique d'aide à la conception. Ces recherches ont aussi montré que les interfaces matérielles étaient étroitement liées aux interfaces logicielles qui les pilotent. La conception ne peut se faire les unes sans les autres pour être vraiment efficace.

Les expérimentations de prototypage d'applications ont mis en évidence quelques difficultés concernant les aspects de communication et les contraintes architecturales de la plateforme qui restreignent la réutilisation des couches basses du logiciel. Les travaux menés ont défini des techniques pour contourner ces difficultés, et il reste encore à développer des outils pour faciliter cette phase de prototypage.

VII.2 Discussion

On peut s'interroger sur le fait qu'aucun outil commercial n'apporte de solutions satisfaisantes de conception de systèmes logiciel/matériel, alors que la recherche dans ce domaine existe depuis le début des années 1990. En fait, une des raisons majeures semble être la confusion qui règne dans

les esprits concernant la conception. En effet, on peut considérer qu'un système monopuce est composé de trois principaux types de composants :

- Les processeurs
- Les mémoires
- Les bus ou NoC ("Network On Chip")

Le flot de conception donné dans la Figure 4 s'applique pour ces trois types de composants, car ils sont tous constitués d'une partie logicielle et d'une partie matérielle. Le processeur exécute un programme, l'architecture mémoire est utilisée par le logiciel applicatif à travers des fonctions d'entrées/sorties logicielles et/ou matérielles. Et les réseaux de communication mettent aussi en œuvre des connexions physiques pilotées par des programmes. De plus, ce même flot s'applique aussi au système global. Il faut donc appliquer ce flot à plusieurs reprises, sur le système puis sur chacun des types de composant du système.

Ce flot de conception nécessite alors quatre types de métiers ou concepteurs : l'architecte, le concepteur de la partie logicielle, le concepteur de la partie matérielle, et l'intégrateur. On peut ajouter un ou plusieurs types de métiers transversaux pour la validation et le test par exemple, mais aussi pour assurer la collaboration entre tous.

Ces quatre types de métiers nécessitent quatre types d'outils d'aide à la conception, qui doivent pouvoir s'appliquer sur les trois types de composants de bases. Et c'est là une des difficultés pour l'obtention d'un outil automatique. De plus, selon que l'on vise la simulation, le prototypage ou la réalisation d'un circuit physique, certaines parties, logicielles ou matérielles peuvent être différentes. On augmente alors le nombre et le type d'objets manipulés. Ce qui est encore amplifié quand on sait que plusieurs niveaux d'abstraction sont toujours manipulés pour cacher les détails d'implémentation.

On comprend alors qu'un seul outil n'est pas une solution réaliste, car la conception d'un système nécessite la collaboration de tous ces métiers pour tous les composants, et les outils de conception ont un champ d'application réduit, et ne prennent pas en compte les autres types de composants.

Une des solutions est de posséder un environnement unique, composé d'une base de données de composants et de modèles, et de méthodes pour assembler et raffiner ces composants. Cet outil doit être capable de manipuler plusieurs niveaux d'abstraction, de viser la simulation, le prototypage ou la réalisation d'un circuit, de proposer une méthode de raffinement pour tous les types de composants. ROSES répond à une partie de ces objectifs, et tous les travaux de l'équipe SLS vise à le compléter dans cet optique.

VII.3 Perspectives et évolution

Les limites physiques de la technologie étant sans cesse repoussées, il n'existe à ce jour aucune raison au ralentissement du développement des systèmes intégrés. Le changement apparu ces dernières années vient de la place prépondérante prise par le logiciel dans de tels systèmes, ce qui n'était pas vrai il y a encore cinq ans. Aussi le développement et la validation de ce logiciel pour l'architecture matérielle sur laquelle il est exécuté sont devenus un nouvel axe de recherche. Il devrait se poursuivre, poussé par les besoins des industriels qui reconnaissent manquer de solutions efficaces.

La problématique de spécification, conception et validation des composants matériels et logiciels ou les relations qui unissent ces composants devrait continuer d'exister car de nombreuses difficultés n'ont pas encore trouvé de solutions. On peut s'attendre dans les prochaines années à plus d'intérêts encore pour l'intégration des systèmes logiciel/matériel.

Une autre problématique devrait continuer de préoccuper les concepteurs de systèmes, c'est l'adéquation entre l'architecture et les différentes couches logicielles, notamment l'application et le système d'exploitation. Il existe des liens et des dépendances forts entre architecture et logiciel, qui sont plutôt bien maîtrisés dans le cas de la compilation, mais qui sont plus difficiles à appréhender entre un système d'exploitation et les ressources physiques d'une architecture complexe d'un système.

C'est à ces deux tâches, intégration logiciel/matériel et adéquation architecture/système d'exploitation que je compte m'attaquer dans la suite de mes travaux de recherche.

VIII Bibliographie

[ABO03] *ABOULHAMID EL M.., A new approach to system-level design*, couverture de EEDesign, 12 janvier 2003,

[ABR02] *ABRIL GARCIA A., GOBERT J., DOMBEK T., MEHREZ H., PÉTROT F., Energy Estimations in High Level Cycle-Accurate Descriptions of Embedded Systems*, The 5th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'2002), Brno, Czech Republic, pp. 228-235, April 2002.

[BAG02] *BAGHDADI A., ZERGAINOH N.E., CESARIO W. JERRAYA A.A., Combining a Performance Estimation Methodology with a Hardware/Software Codesign Flow Supporting Multiprocessor Systems*, IEEE Transactions on Software Engineering, pp. 822-831, vol. 28, numéro 9, sept. 2002.

[BEN02] *BEN CHEHIDA K., AUGUIN M., Hw/Sw Partitioning Approach for Reconfigurable System Design*, International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 247-251, 2002.

[BOI04] *BOIS G., FILION L., TSIKHANOVICH A., ABOULHAMID E. M.*, *Modélisation, raffinement et techniques de programmation orientée objet avec SystemC*, chapitre de l'ouvrage "La spécification et la validation des systèmes hétérogènes embarqués", A. A. Jerraya and G. Nicolescu, Eds.: Hermes, 2004, pp. 171-207.

[BOR98] *BORRIELLO G., LAVAGNO L., ORTEGA R., Interface synthesis: a vertical slice from digital logic to software components*, ICCAD, pp. 693-695, 1998.

[BRU00] *BRUNEL J.Y., KRUIJTZER W., KENTER H., PÉTROT F., L. PASQUIER, DE KOCK E. AND SMITS W., COSY Communication IP's*, DAC 2000.

[CAT02] *CATTHOOR F., Data Access and Storage Management for Embedded Programmable Processors,* Kluwer Academic Publishers, 2002.

[CAT98] *CATTHOOR F., WUYTACK S., GREEF E., BALASA F., NACHTERGAELE L., VANDECAPPELLE A.., Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*, Kluwer Academic Publishers, 1998.

[CES01] *CESARIO W., NICOLESCU G., GAUTHIER L., LYONNARD D., JERRAYA A. A.*, *Colif: a Multi Design Representation for Application-Specific Multiprocessor System-on-Chip Design*, 12th Rapid System Prototyping (RSP), California, 2001.

[CES02] *CESARIO W. BAGHDADI A., GAUTHIER L., LYONNARD D., NICOLESCU G., PAVIOT Y., YOO S., JERRAYA A. A.*, *Component-Based Design Approach for Multicore SoCs*, DAC, New Orleans, juin 2002.

[CHA99] *CHANG H., COOKE L., HUNT M, MARTIN G, MCNELLY A, TODD L., Surviving the SOC Revolution – A Guide to Plateforme-Based Design*, Kluwer Academic Publisher, 1999

[FRA01] *FRABOULET A., KODARY K., MIGNOTTE A., Loop Fusion for Memory Space Optimization*, International Symposium on System Synthesis, Montreal, Canada, Sept-Oct 2001.

[FRA99] *FRABOULET A., HUARD G., MIGNOTTE A.*, *Loop Alignement for Memory Accesses Optimization*, International Symposium on System Synthesis, San Jose, California, USA, Novembre 1999.

[GAJ98] *GAJSKI D. D., VAHID F., NARAYAN S., AND GONG J.,* SpecSyn: *An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design*, IEEE Transactions on VLSI Systems, Vol. 6, No. 1, March 1998.

[GAU01] *LOVIC GAUTHIER, Génération de systèmes d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques,* thèse de Doctorat INPG, Spécialité Microélectronique, laboratoire TIMA, 2001.

[GHA03] *FERID GHARSALLI, Conception des interfaces logiciel-matériel pour l'intégration des mémoires globales dans les systèmes monopuces*, thèse de doctorat, INPG, Spécialité Informatique, laboratoire TIMA, 2003.

[GRA04] *GRASSET A., ROUSSEAU F., JERRAYA A.A., Network Interface Generation for MPSOC: from Communication Service Requirement to RTL Implementation*, 15th Rapid System Prototyping (RSP) workshop, Genève, 2004.

[GRU00] *GRUN P.,DUTT N., NICOLAU A., Memory Aware Compilation Through Accurate Timing Extraction,* DAC, Los Angeles, juin 2000.

[GRU03] *GRUN P.,DUTT N., NICOLAU A., Memory Architecture Exploration for Programmable Embedded Systems*, Hardcover, 2003.

[HAR97] *HARDT W., ROSENTIEL W., Prototyping of Tightly Coupled Hardware/Software System*, ACM Transactions on Design Automation for Embedded Systems, volume 2, pages : 283-317, Kluwer Academic Publisher, 1997.

[HOM01] *HOMMAIS D., PÉTROT F., AUGÉ I., A Practical Toolbox for System Level Communication Synthesis,* Rapid System Prototyping (RSP) workshop, 2001.

[ISM96] *ISMAIL T.B., DAVEAU J., O'BRIEN K., AND JERRAYA A.A., A system-level communication synthesis approach for hardware/software systems*, Microprocessors and Microsystems, 20(3):149-157, May 1996.

[I2O02] Intelligent Inout/Output SIG: http://www.i2osig.org/, 2002.

[JEN97] *D.C.R. JENSEN, .MADSEN J., AND PEDERSEN S.*, *The importance of Interfaces: A HW/SW codesign case study*, Workshop on Codesign, 1997.

[JER02a] *JERRAYA A.A, Conception de haut niveau des systèmes monopuces,* Hermes, Mai 2002.

[JER02b] *JERRAYA A.A, Conception logique et physique des systèmes monopuces,* Hermes, Mai 2002.

[KAT99] *KATAYAMA T., SAISHO K., FUKUDA A., Proposal of a Support System for Device Driver Generation*, Asia-Pacific Software Engineering Conference (APSEC'99), pp.494-497, 1999.

[KNU98] *KNUDSEN P. V., MADSEN J.*, *Integrating Communication Protocol Selection with Hardware/Software Codesign*, International Symposium on System Synthesis (ISSS), 1998.

[LOP03] *LOPEZ-VALLEJO M., LOPEZ J.C., On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques*, ACM Transactions On Design Automation of Electronic Systems (TODAES), vol. 8, n° 3, pp. 269-297, juillet 2003.

[LYO01] *LYONNARD D., YOO S., BAGHDADI A., JERRAYA A. A., Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System–on-Chip*, DAC 2001.

[LYO03] *DAMIEN LYONNARD, Approche d'assemblage systématique d'élément d'interface pour la génération d'architectures multiprocesseurs*, Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2003.

[MEF01] *MEFTALI S., GHARSALLI F., ROUSSEAU F., JERRAYA A.A., An Optimal Memory Allocation for Application-Specific Multiprocessor System-on-Chip,* International Symposium on System Synthesis (ISSS 2001), pp. 19-24, Montréal, Canada, Sept-Oct. 2001.

[MEF02a] *MEFTALI S., GHARSALLI F., ROUSSEAU F., JERRAYA A.A., Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, SoC Design Methodology, pp 193–2004, Kluwer Academic Pulishers, Juin 2002.

[MEF02b] *SAMY MEFTALI, Exploration d'architectures et allocation/affectation mémoire dans les systèmes multiprocesseurs monopuces*, Thèse de doctorat, Université Joseph Fourier - Grenoble, Spécialité Informatique, laboratoire TIMA, 2002.

[MIS01a] *MISHRA P., ROUSSEAU F., DUTT N., NICOLAU A., Architecture Description Language Driven Design Space Exploration in the Presence of Coprocessors*, Workshop on Synthesis And System Integration of MIxed technologies (SASIMI 2001), pp. 67-71, Nara, Japon, Oct. 2001.

[MIS01b] *MISHRA P., DUTT N., NICOLAU A., Functional Abstraction-driven Design Space Exploration of Heterogeneous Programmable Architectures*, International Symposium on System Synthesis (ISSS 2001), pp. 19-24, Montréal, Canada, Sept-Oct. 2001.

[MIS04] *PRABHAT KUMAR MISHRA P., Specification-driven Validation of Programmable Embedded Systems,* Ph.D. dissertation, University of California at Irvine, 2004.

[NIC02] *EUGENIA G. N. NICOLESCU, Spécification et validation des systèmes hétérogènes embarqués*, thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2002.

[NIE98] *NIEMANN R., Hardware/sofware Co-design for Data Flow Dominated Embedded Systems,* Kluwer Academic Publishers, 1998.

[OBE01] *ÖBERG J., O'NILS M., JANTSCH A.,POSTULA A., HEMANI A.: Grammar-based design of embedded systems*, Journal of Systems Architecture, vol. 47, n° 3-4, avril 2001.

[ONI01] *O'NILS M., JANTSCH J., Device Driver and DMA Controller Synthesis from HW/SW Communication Protocol Specifications*, in Design Automation for Embedded Systems, Vol. 6, No. 2, Kluwer Academic Publisher, 2001.

[PAS98] *PASSERONE R., ROWSON J. A., S.-VINCENTELLI A.: Automatic Synthesis of Interfaces between Incompatible Protocols*, DAC 1998.

[PAV04] *YANICK PAVIOT, Implémentations mixtes logicielles/matérielles des services de communication pour l'exploration du partitionnement logiciel/matériel,* thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2004.

[SAS04] *ARIF SASONGKO, Prototypage basée sur une plateforme reconfigurable pour la vérification des systèmes monopuces,* thèse de doctorat, Université Joseph Fourier, Spécialité Microélectronique, laboratoire TIMA, 2004.

[SEA96] *SEAWRIGHT A., HOLTMANN U., MEYER W., PANGRLE B., VERBRUGGHE R., BUCK J.: A System for Compiling and Debugging Structured Data Processing Controllers*, Euro-DAC 1996.

[SIA02] *SIARRY P., Application des métaheuristiques d'optimisation en électronique,* [RE 8], traité recherche, Techniques de l'Ingénieur, 2002.

[SIE02] *SIEGMUND R., MÜLLER D.: Automatic Synthesis of Communication Controller Hardware from Protocol Specifications*, IEEE Design & Test of Computers, vol. 19, n°4, juillet/août 2002.

[SYS00] Synopsys Inc., *SystemC*, available at http://www.systemc.org/.

[VER96] *VERCAUTEREN S., LIN B., MAN H. D., Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications*, DAC 1996.

[ZER02] *ZERGAINOH N. E.., Méthodologie et modèles pour la conception digitale,* chapitre de l'ouvrage "Conception de haut niveau des systèmes monopuces", traité EGEM Electronique, Hermès Science Publications, mai 2002.

[ZIT93] *ZITTERBART M., STILLER B., TANTAWY A. N., A Model for Flexible High-Performance Communication Subsystems,* IEEE Journal on Selected Areas in Communications, vol. 11, n°4, mai 1993.

IX Curriculum vitae

ROUSSEAU Frédéric

Né le 2 mars 1967 à Valence (26), France
Nationalité française, célibataire.

Adresse professionnelle
Laboratoire TIMA, équipe SLS
46 avenue Félix Viallet
38031 GRENOBLE Cedex
tel : (33) 4 76 57 46 41
email : frederic.rousseau@imag.fr

Situation actuelle

Depuis Oct.1999 **Maître de Conférences à l'Université Joseph Fourier – Grenoble**
Enseignement à Polytech'Grenoble, écoles d'ingénieurs de l'Université
Recherche au Laboratoire TIMA (Technique de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs)

Diplômes et formations

Juillet 1997 **Doctorat Informatique, Université d'Evry Val d'Essonne**
Etude du partitionnement logiciel/matériel : application aux systèmes de télécommunications orientés flot de données

Sept. 1992 **DEA Informatique, INP Grenoble**

Sept. 1991 **DEA de Microélectronique, Université Joseph Fourier, Grenoble**

Sept. 1991 **Diplôme d'Ingénieurs 3i (Informatique Industrielle et Instrumentation)** Université de Grenoble (ISTG devenu Polytech'Grenoble)

Expérience professionnelle

Oct. 1996 à Sept. 1999 **Ingénieur Enseignant Chercheur à l'ESIM, Ecole Supérieure d'Ingénieurs de la chambre de Commerce et d'Industrie de Marseille Provence (CCIMP).**
Enseignant – chercheur au département EII (Electronique Informatique Industrielle)
Réalisation de contrats industriels pour les services de la CCIMP.

X Activités d'enseignement

X.1 Enseignement

Mes activités d'enseignement ont réellement commencé à l'Ecole Supérieure d'Ingénieurs de Marseille (ESIM), école d'ingénieurs généralistes de la chambre de commerce et d'industrie de Marseille Provence. Ingénieur-enseignant-chercheur dans le département d'Electronique et d'Informatique Industrielle de l'ESIM d'octobre 1996 à septembre 1999, j'ai enseigné dans les 3 années de formation initiale en cours, TD et TP, l'informatique, l'électronique numérique et analogique, et l'automatique continue. J'ai aussi encadré de nombreux projets d'étude en informatique industrielle avec conception de la carte à microprocesseurs et des programmes.

J'ai été nommé sur un poste de maître de conférences à l'Université Joseph Fourier de Grenoble en octobre 1999, pour rejoindre l'ISTG (Institut des Sciences et Techniques de Grenoble), devenue Polytech'Grenoble et regroupant plusieurs filières d'ingénieurs de l'Université. J'enseigne principalement au département 3i (Informatique Industrielle et Instrumentation), mais aussi en licence d'informatique et master recherche micro-nano électronique.

A 3i, j'interviens en 1ère et 2ième année en programmation (langage C et algorithmique), en projet d'informatique, en système d'exploitation (Linux), en architecture des processeurs (RISC) et les interfaces logiciel/matériel. Une partie de ces enseignements a été mise en place avec le nouveau cursus en 3i, demandé par la commission du titre pour l'habilitation à délivrer le diplôme. J'ai participé activement à la mise en place de cette nouvelle maquette, en redéfinissant les objectifs de l'apprentissage de l'informatique et le découpage des enseignements.

Le premier changement important que j'ai apporté en 2000 est l'utilisation du système d'exploitation Linux, et la gestion d'une salle de 15 machines. Ainsi tout l'apprentissage de la programmation en langage C est fait sous linux. Les projets d'informatique sont aussi une activité nouvelle. A titre d'exemple, des projets tels que des assembleurs (68HC11, MIPS R3000), simulateurs ou algorithmes de compression (Huffman, Shanon-Fado) sont placés en fin de 1ère année et utilisent les connaissance acquises tout au long de l'année.

En master recherche Micro-Nano Electronique, j'ai conçu un cours (et TD) de conception de systèmes sur puce. Il s'agit de présenter les règles et les principes de conception de systèmes numériques, mais aussi d'aborder certains points clés des systèmes monopuces que sont les aspects architecture des processeurs, les caches et les mémoires, et le découpage logiciel/matériel.

J'enseigne aussi l'architecture logiciel/matériel en licence d'informatique et l'électronique analogique (cours, TD et TP créés en septembre 1999 pour le nouveau département RICM de Polytech'Grenoble).

X.2 Activités et responsabilités administratives et collectives

- Responsable pédagogique de la 1ère année 3i (Polytech'Grenoble) depuis septembre 2000 (entre 55 et 65 étudiants).
- Coadministrateur d'une salle Linux pour l'enseignement à Polytech'Grenoble depuis septembre 2000.
- Responsabilités traditionnelles de suivi de stages et de projets d'élèves ingénieurs.

- Participation au processus de recrutement des élèves ingénieurs (Jurys de DUT, étude des dossiers, entretiens) à Polytech'Grenoble.
- Tuteurs d'étudiants apprentis – ingénieurs (formation par apprentissage pour certains étudiants de 3i).

Avant mon affectation au TIMA (activités et responsabilités à l'ESIM)

- Responsable de l'option "Electronique et Informatique Industrielle" en 2ième et 3ième années à l'ESIM pour l'année 1998/1999.
- Responsable de l'option "Conception Microélectronique" en 3ième année de l'école ISMEA (Institut Supérieur de Micro-Electronique Appliquée), école d'ingénieurs du groupe ESIM pour l'année 1998/1999.
- Tuteurs d'étudiants apprentis – ingénieurs (formation par apprentissage pour certains étudiants à l'ESIM).

XI Activités de recherche

XI.1 Invitations par des laboratoires étrangers

- Invitation pour un séjour de 4 mois (mai – sept. 2000) à l'Université d'Irvine (University of California at Irvine) dans le laboratoire "Center for Embedded Computer Systems" sous la direction des profs. D. Gajski et N. Dutt. Le travail concernait l'étude et la comparaison de différentes solutions architecturales de l'organisation mémoire pour l'intégration d'un coprocesseur dans un cœur de microprocesseur VLIW.
- Invitation de deux semaines à l'Université de Montréal (oct. 2004). Présentation de mes travaux, rencontre et discussions avec les chercheurs de l'Université et ceux de Polytechnique Montréal. Participation à un jury de thèse.

XI.2 Communication de la recherche

- Relecteur de plusieurs revues scientifiques : TECS (ACM on Transactions on Embedded Computing Systems) et Journal "Formal Methods in System Design" (Kluwer)
- Participation au "technical program committee" de la conférence DATE (depuis 2002) sur le thème "Hardware/software Codesign".
- Membre du "program committee" du Workshop on Rapid System Prototyping depuis 2003.
- "Finance chair" de HLDVT (IEEE International High Level Design Validation and Test Workshop), oct. 2002 à Cannes (65 participants).
- "Chairman" de plusieurs sessions de conférences (DATE, ISSS, RSP).
- Participation aux Actions Spécifiques STIC du CNRS : J'ai participé activement pendant un an à l'action spécifique : Systèmes d'exploitation et Architectures Multiprocesseurs des Systèmes Complexes intégrés sur puce.
- Relecteur de plusieurs conférences ou "workshops" : DAC, DATE, RSP.

XI.3 Participation à des jurys de thèse

- Luc Charest, Université de Montréal (rapporteur externe), *De la fusion du génie logiciel et d'une bibliothèque à source ouverte pour la modélisation/simulation de processus matériel et logiciel*, octobre 2004.
- Arif Sasongko, Université Joseph Fourier (examinateur), *Prototypage basé sur une plateforme reconfigurable pour la vérification des systèmes monopuces*, octobre 2004.
- Ferid Gharsalli, INPG (examinateur), *Conception des interfaces logiciel/matériel pour l'intégration des mémoires globales dans les systèmes monopuces*, juillet 2003.
- Samy Meftali, Université Joseph Fourier (examinateur), *Exploration d'architectures et allocation/affectation mémoire dans les systèmes multiprocesseurs monopuces*, septembre 2002.
- Nourredine Chabini, Université de Montréal (rapporteur externe), *Méthode pour améliorer la qualité des implantations matérielles de systèmes informatiques*, octobre 2001.

- Guy Gogniat, Université de Nice – Sophia Antipolis (examinateur), *Architecture générique et synthèse des communications pour la conception conjointe de systèmes embarqués logiciel/matériel*, novembre 1997.

XI.4 Participation à des contrats industriels ou européens

- Coordinateur d'un projet industriel avec la société MnD : Développement d'une application d'encodage vidéo sur un système embarqué multiprocesseur (janvier 2005 – juin 2005).

  Il s'agit de porter une application d'encodage vidéo sur une plateforme basée sur un système multiprocesseur monopuce (4 processeurs SPARC V8), en utilisant les techniques de génération des couches logicielles bas niveau développées dans l'équipe SLS du TIMA.

- Participation au projet européen Trust-Es (juillet 2004 – arrêté en février 2005).

  De nombreuses sociétés, notamment ST, ATMEL, THALES, IRoC, … s'intéressent à la sécurité des cartes à puces. La participation de l'équipe SLS du TIMA se limitait à l'étude des problèmes liés aux systèmes d'exploitation dans les terminaux lecteurs, et à l'aspect sécurité dans la reconfiguration de ces mêmes lecteurs.

- Participation au projet ARCHIFLEX (mars – novembre 2003).

  Mise en œuvre avec ST Microelectronics et le LETI (CEA) d'un environnement de prototypage d'applications multiprocesseurs, autour des processeurs ST, ARM, et du réseau de communication développé au LETI. Définition des besoins en communications et des différentes couches logicielles de bas niveau.

Avant mon affectation au TIMA (contrats réalisés à l'ESIM)

- Contrat de développement d'une carte d'acquisition de données physiologiques pour une PME locale (Fév. - Juil. 1999).

  Réalisation d'un électromyogramme (mesure de l'intensité de l'activité musculaire), en respectant les normes de sécurité. Carte d'acquisition à base d'un microprocesseur 80C552 et de différents composants d'instrumentation (amplificateur d'instrumentation, amplificateur d'isolement, filtre, …).

- Participation au projet européen AORTICS (Advanced Open Resources Telematics In Critical Care Situations) (Mai – Nov. 1998).

  Développement de l'interface graphique permettant la visualisation de signaux physiologiques (cardiaque, neurologique, et respiratoire) en VisualC++ sous WindowsNT.

- Réalisation pour EDF-Centre de Calcul de Marseille de deux projets (de Nov. 1996 à Juil. 1997) : Modélisation d'un réseau de moteurs asynchrones et simulation en régime transitoire et Simulation du comportement d'un transformateur triphasé en régime transitoire.

XI.5 Autres

- Membre titulaire de la commission de spécialiste, 27ième section, à l'université d'Evry Val d'Essonne en tant que membre extérieur depuis oct. 2001.

XII Encadrement de thèse ou de DEA (master recherche)

XII.1 Encadrements de thèses

Depuis 1999, je co-encadre des étudiants en thèse sur les thèmes liés à la conception des systèmes monopuces. 3 thèses ont déjà été soutenues, 2 sont en cours.

Samy MEFTALI	***Années de thèse :** 1999 - 2002*
Titre : EXPLORATION D'ARCHITECTURALE ET ALLOCATION/AFFECTATION MEMOIRE DANS LES SYSTEMES MULTIPROCESSEURS MONOPUCES	
Etat : soutenue (6 septembre 2002)	
Financement: ministère	
Encadrement : A. Jerraya (20%), F. Rousseau (80%)	
Devenu : Maître de Conférences à Lille - LIFL	

Ferid Gharsalli	***Années de thèse :** 2000 - 2003*
Titre : CONCEPTION DES INTERFACES LOGICIEL/MATERIEL POUR L'INTEGRATION DES MEMOIRES GLOBALES DANS LES SYSTEMES MONOPUCES	
Etat : soutenue (1er juillet 2003)	
Financement: ministère	
Encadrement : A. Jerraya (10%), F. Rousseau (90%)	
Devenu : En post-doc aux Etats-Unis	

Arif Sasongko	***Années de thèse :** 2001 - 2004*
Titre : PROTOTYPAGE BASE SUR UNE PLATEFORME RECONFIGURABLE POUR LA VERIFICATION DES SYSTEMES MONOPUCES	
Etat : soutenue (15 octobre 2004)	
Financement: contrat	
Encadrement : A. Jerraya (10%), F. Rousseau (90%)	
Devenu : Sous contrat au TIMA	

Arnaud Grasset	***Années de thèse :** 2002 - 2005*
Titre : SYNTHESE DES INTERFACES MATERIELLES DANS LA CONCEPTION DES SYSTEMES MONOPUCES : DE LA SPECIFICATION A LA GENERATION AUTOMATIQUE	
Etat : en cours	
Financement: ministère	
Encadrement : A. Jerraya (10%), F. Rousseau (90%)	

Benaoumeur Senouci	***Années de thèse :*** *2004 - 2007*
Titre : AUTOMATISATION DU FLOT DE PROTOTYPAGE BASE SUR UNE PLATEFORME RECONFIGURABLE POUR LA VALIDATION DES INTERFACES LOGICIEL/MATERIEL	
Etat : en cours	
Financement: ministère	
Encadrement : F. Pétrot, F. Rousseau	

XII.2 Encadrement de DEA (master recherche)

Ferid GHARSALLI, *Conception mixte logiciel/materiel des systèmes multiprocesseurs avec mémoire*, DEA d'Informatique : Systèmes et communications, INPG, juin 2000.

Benaoumeur SENOUCI, *Etude de la génération des architectures locales dans un système multiprocesseur monopuce en utilisant le flot ROSES*, master recherche Micro-Nano Electronique, Université Joseph Fourier – Grenoble, juin 2004.

Abdelmajid BOUAJILA, Prototypage rapide d'architectures multiprocesseur à base de processeurs flexibles, master recherche Micro-Nano Electronique, Université Joseph Fourier – Grenoble, juin 2005.

XIII Liste des publications

XIII.1 Thèse soutenue

[T1] F. Rousseau, *Etude du partitionnement logiciel/matériel : application aux systèmes de télécommunication orientes flot de données*, Thèse de doctorat, Université d'Evry Val d'Essonne, juillet 1997.

XIII.2 Chapitres d'ouvrage

[O1] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya, *Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, chapitre de l'ouvrage : SoC Design Methodologies, Kluwer Academic Publishers, juin 2002, ISBN 1-4020-7148-5.

[O2] F. Rousseau, *La conception système et le partitionnement logiciel/matériel*, chapitre de l'ouvrage : Conception de haut niveau des systèmes monopuces, Traité EGEM Electronique, Hermès Science Publications, mai 2002, ISBN 2-7462-0433-9.

[O3] M. Aiguier, J. Benzakki, G. Bernot, S. Beroff, D. Dupont, L. Freund, M. Israel, F. Rousseau, *ECOS: Environnement générique de cospécification*, chapitre de l'ouvrage : CODESIGN conception conjointe logiciel-matériel, C.T.I. COMETE, Eyrolles, 1998, ISBN 2-2120-5219-7.

[O4] M. Aiguier, J. Benzakki, G. Bernot, S. Beroff, D. Dupont, L. Freund, M. Israel, F. Rousseau, *ECOS: A Generic Codesign Environment for the Prototyping of Real Time Applications "From Formal Specifications to Hardware/Software Partitioning"*, chapitre de l'ouvrage : Hardware/Software Co-Design and Co-Verification, Current Issues in Electronic modeling, N° 8, pp. 23 - 57, Kluwer AP, 1997, ISBN 0-7923-9689-8.

XIII.3 Revue internationale avec comité de lecture

[RI1] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *Towards SoC Validation Through Prototyping: A Systematic Approach Based on Reconfigurable Platform*, Design Automation for Embedded Systems, vol. 8, pp 155-171, Kluwer Academic Publishers, déc. 2003.

[RI2] L. Freund, M. Israel, F. Rousseau, J. M. Berge, M. Auguin, C. Belleudy, G. Gogniat, *A codesign experience in acoustic echo cancellation: GMDFα*, ACM Transactions on Design Automation of Embedded Systems, vol. 2, N. 4, octobre 1997.

XIII.4 Revue industrielle

[R1] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *SoC Validation trough Prototyping on ARM Integrator Platform*, ARM Information Quarterly (IQ), volume 2, numéro 2, 2003.

XIII.5 Revue nationale

[RN1] F. Gharsalli, F. Rousseau, A. Jerraya, *Conception des interfaces logiciel-matériel pour l'intégration des mémoires globales dans les systèmes monopuces*, Techniques et Sciences Informatiques, à paraître en 2005.

[RN2] F. Rousseau, *Conception des Systèmes VLSI*, E 2 455, ouvrage de base édité par les Techniques de l'Ingénieur, fév. 2005.

XIII.6 Conférence internationale avec comité de lecture et actes

[CI1] F. Rousseau, A. Sasongko, A. Jerraya, *Shortening SoC Design Time with New Prototyping Flow on Reconfigurable Platform*, NorthEast Workshop on Circuits And Systems (NEWCAS 2005), Québec, Canada, juin 2005.

[CI2] A. Grasset, F. Rousseau, A. Jerraya, *Automatic Generation of Component Wrappers by Composition of* , Workshop on Rapid System Prototyping (RSP 2005), Montréal, Canada, juin 2005.

[CI3] A. Grasset, F. Rousseau, A. Jerraya, *Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation*, Workshop on Rapid System Prototyping (RSP 2004), pp. 66-69, Genève, Suisse, juin 2004.

[CI4] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *Embedded Application Prototyping on a Communication-Restricted Reconfigurable Platform*, Workshop on Rapid System Prototyping (RSP 2003), pp. 33-39, San Diego, EU, juin 2003.

[CI5] F. Gharsalli, S. Meftali, F. Rousseau, A. Jerraya, *Unifying Memory and Processor Wrapper Architecture in Multiprocessor SoC Design*, International Symposium on System Synthesis (ISSS 2002), pp. 26-31, Kyoto, Japon, oct. 2002.

[CI6] F. Gharsalli, S. Meftali, F. Rousseau, A. A. Jerraya, *Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC*, Design Automation Conference (DAC 2002), pp. 596-601, La nouvelle Orléans, EU, juin 2002.

[CI7] S. Meftali, F. Gharsalli, F. Rousseau, A. A. Jerraya, *Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, International Conference on Very Large Scale Integration: the global System on Chip Design (VLSI-SoC 2001), pp. 17-22, Montpellier, France, déc. 2001.

[CI8] P. Mishra, F. Rousseau, N. Dutt, A. Nicolau, *Architecture Description Language Driven Design Space Exploration in the Presence of Coprocessors*, Workshop on Synthesis And System Integration of MIxed technologies (SASIMI 2001), pp. 67-71, Nara, Japon, oct. 2001.

[CI9] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya, *An Optimal Memory Allocation for Application-Specific Multiprocessor System-on-Chip*, International Symposium on System Synthesis (ISSS 2001), pp. 19-24, Montréal, Canada, sept-oct. 2001.

[CI10] L. Freund, D. Dupont, M. Israel, F. Rousseau, *Overview of the ECOS Project*, Workshop on Rapid System Prototyping (RSP 1997), pp. 39-43, Chapel Hill, NC, USA, juin 1997.

[CI11] L. Freund, D. Dupont, M. Israel, F. Rousseau, *Interface Optimization during Hardware-Software Partitioning*, Workshop on Hardware/Software Co-Design (CODES/CASHES 1997), Brunshweig, Allemagne, mars 1997.

[CI12] L. Freund, M. Israel, F. Rousseau, J-M. Berge, M. Auguin, C. Belleudy, G. Gogniat, *A Codesign Experiment in Acoustic Echo Cancellation: GMDFα*, International Symposium on System Synthesis (ISSS 1996), pp. 83 – 89, La Jolla, Ca, EU, novembre 1996.

[CI13] F. Rousseau, J.-M. Berge, M. Israel, *Hardware/Software Partitioning for Telecommunication Systems*, Computer Software and Application Conference (COMPSAC 1996), pp. 484 – 489, Séoul, Corée du Sud, août 1996.

[CI14] F. Rousseau, J. Benzakki, J.-M. Berge, M. Israel, *Adaptation of Force-Directed Scheduling Algorithm for Hardware/Software Partitioning*, Workshop on Rapid System Prototyping (RSP 1995), pp. 33 – 38, Chapel Hill, NC, USA, juin 1995.

XIII.7 Conférence nationale avec comité de lecture et acte

[CN1] Grasset, F. Rousseau, A. Jerraya, *Vers l'Automatisation de la Conception des Coprocesseurs de Communication pour les Systèmes Monopuces*, Journées nationales du réseau doctoral en Microélectronique (JNRDM 05), Marseille, mai 2005.

[CN2] Grasset, F. Rousseau, A. Jerraya, *Génération des Interfaces de Communication pour Systèmes Multiprocesseurs Monopuces : de la Spécification des Services de Communication vers l'Implémentation RTL*, Journées nationales du réseau doctoral en Microélectronique (JNRDM 04), Marseille, mai 2004.

[CN3] F. Gharsalli, S. Meftali, F. Rousseau, A. A. Jerraya, *Générateur d'adaptateurs mémoire pour les architectures multiprocesseurs monopuces*, Colloque CAO (organisé par le CNRS), Paris, mai 2002.

[CN4] L. Freund, M. Israel, F. Rousseau, J-M. Berge, M. Auguin, C. Belleudy, G. Gogniat, *Etude de la conception logiciel/matériel d'une application d'annulation d'écho acoustique*, Colloques CAO de circuits intégrés et systèmes, Grenoble, janvier 1997.

[CN5] F. Rousseau, J-M. Berge, M. Israel, *Synthèse des méthodes et algorithmes de partitionnement logiciel/matériel*, Symposium Architectures Nouvelles de Machines, Rennes, février 1996.

XIII.8 Rapport technique

[RT1] P. Mishra, F. Rousseau, N. Dutt, A. Nicolau, *Coprocessor Codesign for Programmable Architecture*, Technical Report, UCI-ICS 01-13, University of California at Irvine, 2001.

XIV Fiche de synthèse

Activités de recherche

- Co-encadrement de 5 thèses (3 soutenues, 2 en cours)
- Encadrement de 3 stages de Master Recherche
- Publications
    - 4 chapitres de livres
    - 2 articles de revues internationales
    - 1 article de revue industrielle internationale
    - 2 articles de revues nationales
    - 14 articles de conférences internationales
    - 5 articles de conférences nationales

Responsabilités liées à la recherche

- Membre de la commission de spécialistes, 27ième section, de l'Université d'Evry Val d'Essonne depuis 2001.
- Membre de la commission mixte de spécialistes à Polytech Grenoble en 2005.

Activités d'enseignement

- Enseignements à Polytech Grenoble
    - Cours, TD et TP de programmation C
    - TD et TP de système d'exploitation Linux
    - Cours, TD et TP d'architecture logiciel/matériel autour du processeur MIPS
    - Cours, TD et TP d'électronique analogique
- Enseignements en Master recherche
    - Cours et TD de conception des systèmes numériques
- Enseignements en licence d'informatique (L3)
    - TD d'architecture logiciel/matériel (processeur ARM7)

Responsabilités liées à l'enseignement

- Responsable pédagogique de la 1ère année 3i à Polytech Grenoble

XV Recueil des principales publications

[RN2] F. Rousseau, *Conception des Systèmes VLSI*, E 2 455, ouvrage de base édité par les Techniques de l'Ingénieurs, fév. 2005.

[O2] F. Rousseau, *La conception système et le partitionnement logiciel/matériel*, chapitre de l'ouvrage : Conception de haut niveau des systèmes monopuces, Traité EGEM Electronique, Hermès Science Publications, mai 2002, ISBN 2-7462-0433-9.

[O1] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya, *Automatic Code-Transformation and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory*, chapitre de l'ouvrage : SoC Design Methodologies, Kluwer Academic Publishers, juin 2002, ISBN 1-4020-7148-5.

[CI6] F. Gharsalli, S. Meftali, F. Rousseau, A. A. Jerraya, *Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC*, Design Automation Conference (DAC 2002), pp. 596-601, La nouvelle Orléans, EU, juin 2002.

[RI1] A. Sasongko, A. Baghdadi, F. Rousseau, A. Jerraya, *Towards SoC Validation Through Prototyping: A Systematic Approach Based on Reconfigurable Platform*, Design Automation for Embedded Systems, vol. 8, pp 155-171, Kluwer Academic Publishers, déc. 2003.