Approche d'assemblage systématique d'éléments d'interface pour la génération d'architecture multiprocesseur = An approach for the systematic gathering of interface items toward the generation of multiprocessor architectures

D. Lyonnard

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
/_/_/_/_/_/_/_/_/_/_/

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

préparée au laboratoire TIMA dans le cadre de
l'École Doctorale d'Électronique, d'Électrotechnique, d'Automatique et de Traitement du Signal
présentée et soutenue publiquement
par

Damien Lyonnard

le 30 Avril 2003

Titre :

Approche d'assemblage systématique d'éléments d'interface pour la génération d'architecture multiprocesseur

Directeur de thèse : Ahmed Amine JERRAYA

Jury

M. Pierre GENTIL	Président
M. Amara AMARA	Rapporteur
M. Flavio Rech WAGNER	Rapporteur
M. Ahmed Amine JERRAYA	Directeur de Thèse
M. Mario DIAZ-NAVA	Examinateur

Remerciements

Je voudrais profiter de cet espace de liberté, que le protocole offre à l'auteur, pour sincèrement remercier toutes les personnes qui ont plus ou moins directement contribué à l'élaboration de cette thèse.

Ainsi, mes premiers remerciements sont adressés à monsieur Pierre GENTIL, directeur de l'École Doctorale d'Électronique, d'Électrotechnique, d'Automatique et de Traitement du Signal pour avoir sut me conseiller lors mon engagement dans cette quête initiatique qu'est une thèse et pour l'honneur incommensurable qu'il m'octroya en présidant mon jury.

Que messieurs Amara AMARA, professeur à l'Institut Supérieur d'Électronique de Paris et Flavio Rech WAGNER, professeur à l'Universidade Federal do Rio Grande do Sul reçoivent, ici, l'expression de mon meilleur respect pour leur analyse avisée de mes travaux.

Tant pour ses conseils en vue de l'amélioration du manuscrit, que pour les nombreuses opportunités de « palper » la réalité industrielle qu'il m'a offert, j'assure monsieur Mario DIAZ-NAVA de mon éternelle reconnaissance.

Je remercie monsieur Bernard COURTOIS de m'avoir accueilli au sein du laboratoire TIMA, où toutes les conditions étaient réunies pour favoriser mon épanouissement tant scientifique que social.

Je ne disposerai jamais d'assez de place ni des mots justes pour exprimer mon respect et ma gratitude pour monsieur Ahmed Amine JERRAYA. Car non content de m'avoir offert l'opportunité de travailler au sein d'une équipe aussi dynamique qu'ambitieuse, sur un sujet aussi motivant que novateur, d'avoir eu la patience de me diriger au quotidien malgré ma provocante excentricité, il n'a cessé d'agir pour mon intérêt. Aussi promets-je que chacun de nos futurs contacts sera prétexte au témoignage de ma reconnaissance.

Il me serait douloureux de ne point avoir un mot amical aux quelques personnes qui dans l'ombre ont, au quotidien, cimenter les bases du « colosse aux pieds d'argile ».

J.-B. (Christelle, soigne le bien, il est si fragile et c'est le dernier fan de Michel ;) pour ses crises de nerfs si bénéfi ques à mon moral (Si un gars aussi soupe-au-lait que lui a pu trouver une aussi jolie copine, alors pourquoi pas moi ?). La grande star Manolo del photos (the X machine, freine mimosas !), que j'ai connu tout timide et qui maintenant fait les couvertures de « Vogue », « Voici » et « Jeunes et jolies ». Yanick (le furet du Joe's Bar Team) et sa « Mimine », ne changez rien, vous êtes craquants.., Dhanistha (**D.** la vraie, l'unique, l'inimitable) pour nos confi dences et ses soirées de déprimes passées à la relecture de mon manuscrit. J.Q. (alias François Perrin) qui a su trouver, trois ans durant, les ressources nécessaires pour nous pondre une gaffe hebdomadaire. Ghislain (du « Magic System ») dieu du zouk et du jus de fruit. Kamel Gibson (Vainqueur de la Star-Ac.'03) pour ses cascades à skis (avec ou sans Wiss'). Une pensée amicale pour Patrick (le nabo minable gnome des neiges :o) dont j'ai souvent railler le grand âge par jalousie d'une jeunesse aussi bien conservée. Je dois aussi une virile accolade à Greg (fabricant d'allumette chez Rossignol). Merci à João Léonardo Santana et à sa douce Mariana de m'avoir démontré que le Brésil n'est pas un pays peuplé de footballeurs et de travestis. Merci à Feeeeu et à Philou auxquels j'ai servi de cobaye pour expérimenter la théorie de Darwin sur mes neurones.

Que Sonja (mon fl amand rose) reçoive en échange de sa gaieté, de sa dévotion, de sa complicité, de sa blanche innocence et de sa rouge passion, mon sincère respect et mon Oedipienne reconnaissance.

Anh-Vu, Adel, Christian, Lionel, Marius et tous les anciens du CSI, toujours promptes à me soutenir quand la galère essuyait des tempêtes.

Lovic (le « I.P. » psychédélique), Wander (mon « margeritas heroe », « Viva, Las-Vegas »), Sungjoo (maître Yooda), ils m'ont apporté trois ingrédients nécessaires à la confection d'un chercheur : 1^{o} l'originalité, 2^{o} l'analyse et 3^{o} l'abstraction.

Je remercie aussi Gabrièla et Bogdan, Patrice et Dorothé pour leurs contacts chaleureux et réconfortants, mais aussi pour leur expédition de reconnaissance en ces lointaines contrées canadiennes.

Je (ré-)itère ma complète allégeance à ma Princesse Lobna, à qui je promets écoute et réconfort éternels. Iuli, merci de m'avoir offert un peu de fraîcheur, et de m'avoir laissé croire quelques instants, que je n'étais pas encore un vieux cheval. Merci à mes deux phares d'Alexandrie Amel et Faïza pour avoir éclairé de leurs sourires, ma route lorsque je naviguais dans les fbts les plus sombres.

Merci aux MCF (Paul, Nacer et Frédéric) et aux trois générations de thésards du groupe SLS (Fabiano, Philippe, Pascal, Zoltan, Amer. Samy, Férid, Aimen, Wassim, Ludo, Fredo, Arif, Arnaud, Anouard, Adriano, ...) que j'ai eu l'honneur de côtoyer.

Que les membres du groupe CIS soient aussi remerciés (Marc, Alain, Laurent, Gilles, Antoine, Estelle, Fabien, Mohammed, Salim, etc...), je n'ai pas toujours été d'un voisinage agréable.

Tous ceux qui ont subi mon joug, à l'occasion d'un stage (Matthieu, Jérôme, Olivier, Michaël et Xavier, Cédric, Youngchoul, Anis) ou lors de pénibles séances de cours, TD et TP (les promos ERG 2A {00-01, 01-02 et 02-03}, ERG 3A {01-02 et 02-03}, ISTG RICM 2A {00-01 et 01-02}, DEUG SM 2A {99-00 et 00-01}).

Que toutes celles qui consciemment ou inconsciemment ont pris des petits bouts de moi (Patty, Steph., Valérie, Karine, Clarisse et quelques collègues dont je préfère taire les noms), qu'elles aient partagées ou non mes « ambitions », soient remerciées pour ce qu'elles m'ont donné en échange.

Enfin, je tiens à remercier tous les membres de la famille *Equus* pour leurs innombrables soutiens moraux. En effet, *« En avant, calme et droit. »* est une expression récurrente en *Equitare*, mais trouve aussi son application durant l'accomplissement d'une thèse. Que le souvenir de LaGuerrinière soit respectueusement honoré, son *« Demander peu, mais exiger souvent. »* sut quotidiennement cadencer mon travail.

À mon papa et à ma maman...
À mon petit frère...
À Quickly, mon cœur est un bien petit pâturage...

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Problématique de la conception de systèmes électroniques multiprocesseurs et monopuces

Sommaire

« Être capable de passer devant en criant : « Suivez-moi ! » ou « Faites comme moi ! » est aussi insuffisant que de se faire tuer sur le champs de bataille à la tête d'une troupe en désordre et ignorante des règles du combat. Il faut savoir dire avec ordre et méthode ce que vous faîtes et comment vos élèves doivent faire. Pour celà, il faut avoir étudié, il faut augmenter sans cesse vos connaissances techniques. »

– Commandant WALLON.

1.1 Contexte : Évolution des applications microélectroniques vers les systèmes multiprocesseurs monopuces.

Cette thèse a pour cadre la conception des systèmes électroniques embarqués spécifiques. C'est-à-dire les systèmes électroniques ayant des fonctionnalités spécialement adaptées pour obtenir les meilleurs performances des applications dans lesquelles ils sont employés.
Dans ces systèmes, le choix et la mise en œuvre des composantes matérielles prennent une place de plus en plus importante, et leur complexité augmente d'autant : il n'est pas rare de nos jours de rencontrer des systèmes avec plusieurs processeurs de types différents, chacun devant effectuer un grand nombre de communications variées. Pour garantir le bon fonctionnement d'un système global, les éléments de son architecture devront s'échanger des informations. Ces échanges sont désignés par le terme « **communication**s ». Pour prendre en charge le transport de ces communications entre les différents composants, des réseaux d'interconnexions sont couramment utilisés.
Ces architectures peuvent être représentées en plusieurs couches, comme l'illustre la figure 1.1. La couche la plus abstraite est la couche de l'application logicielle, elle communique avec une couche logicielle de plus bas niveau qui représente le système d'exploitation. Ensuite, viennent les composants matériels, puis le réseau de communication matériel.

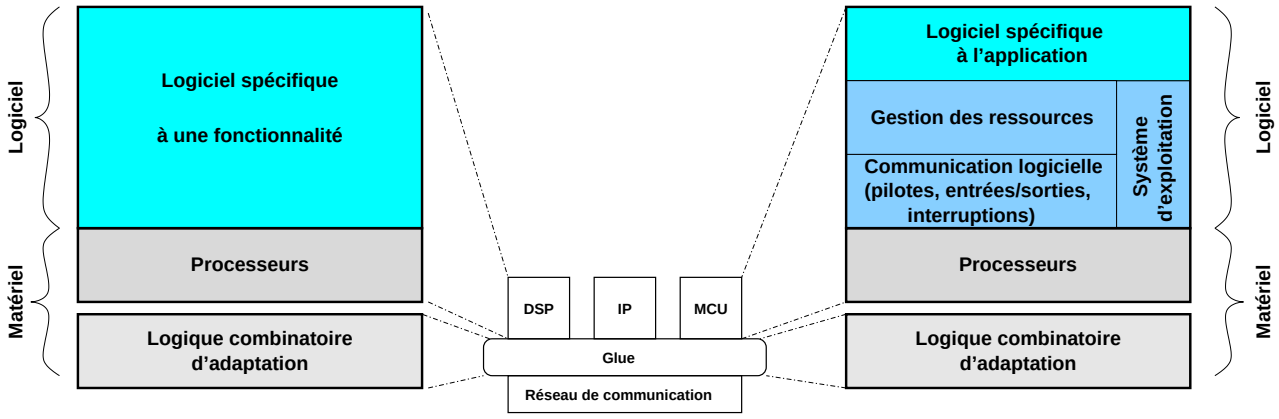


FIG. 1.1 – Représentation des couches architecturales d'un système embarqué

1.2 Accélérer la conception des systèmes multiprocesseurs monopuces

Les systèmes embarqués spécifiques étant très largement utilisés dans les applications récentes dont les fenêtres commerciales sont de plus en plus étroites. Il est donc important de pouvoir les développer rapidement. Cependant, de nombreuses difficultés se présentent lorsqu'il s'agit de développer de tels systèmes.

1.2.1 Motivations : Difficultés de la conception de systèmes multiprocesseurs monopuces

La première difficulté rencontrée par les concepteurs est la complexité des systèmes actuels qui doivent fournir des fonctionnalités de plus en plus élaborées, et peuvent donc être constitués de composants logiciels et matériels variés. Tout ceci aboutit à un volume de spécification des systèmes qui ne peut plus être traité par une équipe traditionnelle de concepteurs, en des délais acceptables aux vues de la concurrence. Les composants réutilisables permettent d'éviter aux concepteurs d'avoir à développer à chaque fois toutes les fonctionnalités requises. Leur composition permet d'obtenir le système complet. Cependant, pour qu'il y ait un gain de temps, l'intégration de ces composants au reste du système ne doit pas demander trop d'efforts aux concepteurs. La variété des composants à mettre en œuvre nécessite des compétences multi-disciplinaires. L'association de plusieurs équipes de concepteurs ayant chacune son domaine d'expertise est une solution. Mais encore faut-il pouvoir autoriser une conception concurrente du logiciel et du matériel, et en garantir à tout moment la consistance et la cohérence. Une autre voie pour surmonter la complexité consiste à élever le plus possible le niveau d'abstraction des descriptions des systèmes à concevoir. Cette méthode, orthogonalement utilisable avec la réutilisation, peut donner aux concepteurs une vision plus générale et plus simple

des systèmes. Des méthodologies et des outils sont alors nécessaires pour guider, accélérer et sécuriser le passage des descriptions de haut niveau aux réalisations finales.

1.2.2 La conception des interfaces entre les composants

Pour que le système fonctionne, les divers composants logiciels et matériels doivent être interconnectés. Il est rare que les composants soient nativement compatibles et puissent être directement reliés entre eux. Il est donc nécessaire de développer des interfaces qui assurent une bonne efficacité des communications entre ces composants. La conception de ces interfaces est la clef de l'étape d'intégration des divers composants du système. C'est un travail long, fastidieux du fait de la très grande diversité des interfaces à créer et sujet à un fort risque d'erreurs. Cette adaptation d'interfaces est décomposable en deux parties :

1. Une partie embarquée dans le composant, pouvant être de nature logicielle (pilote de périphérique) ou matérielle (unité de communication).
2. Une partie externe et matérielle insérée entre les interlocuteurs d'une même communication lorsque leur partie interne ne sont pas suffisamment souples pour correspondre.

Elles sont, toutes deux, spécifiques aux processeurs et plus généralement aux architectures : elles doivent donc être développées en partie ou en totalité pour chaque nouvelle topologie d'interconnexion ou nouveau type d'application.

1.2.3 Vérification des systèmes embarqués spécifiques

La complexité et la diversité des composants des systèmes embarqués spécifiques rendent la validation du système complet très complexe. Les validations formelles requièrent souvent des modèles trop distincts, et des temps de calculs trop importants pour être utilisés pour un système complet, tandis que les validations par simulation ne peuvent pas être exhaustives. Très souvent, faute de modèle de simulation de haut niveau, ces validations générales ne sont effectuées qu'au niveau du prototype. En cas de problème, il est souvent nécessaire d'opérer des itérations dans la conception qui occasionnent des coûts et des délais de développement très importants.

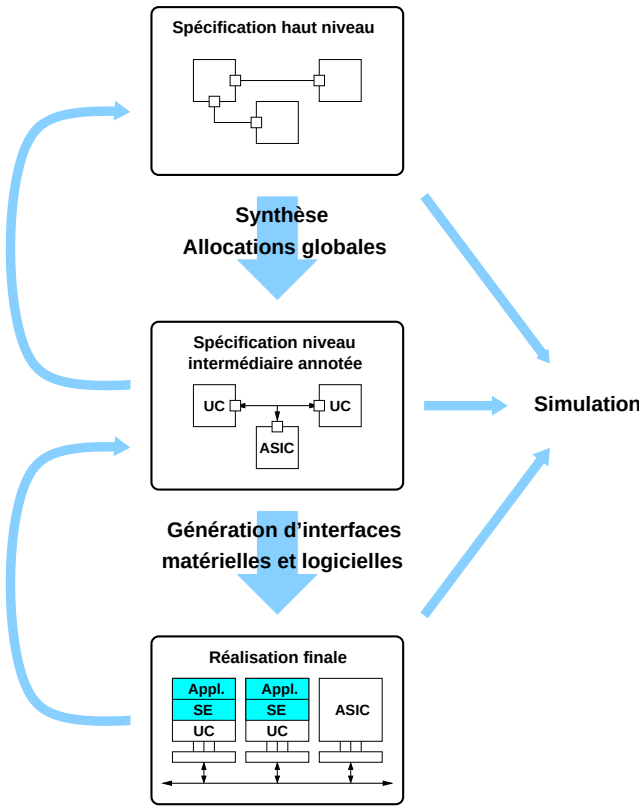


FIG. 1.2 – Un flot de conception pour les systèmes embarqués spécifiques.

1.2.4 Vers une automatisation plus importante de la conception

La conception des systèmes embarqués spécifiques précédemment introduite, est une tâche de plus en plus difficile. Cette difficulté est de nos jours si importante, que ces systèmes sont plus limités par les méthodes et les outils de conception actuels que par la technologie. L'objectif général encadrant cette thèse est de proposer un flot de conception pour les systèmes embarqués spécifiques qui faciliterait le travail des concepteurs. Ce flot traite des architectures dans lesquelles les processeurs sont traités comme des composants destinés à exécuter des fonctions bien définies. Il se concentre sur les aspects de communication et sur la conception des interfaces. Plus précisément, ce flot propose :

– de décrire le système à un niveau d'abstraction élevé ;
– d'automatiser le plus possible les passages aux niveaux d'abstraction inférieurs, jusqu'à la réalisation finale ;

– de pouvoir simuler l'ensemble du système à chaque niveau d'abstraction intervenant dans une approche de conception par raffinements successifs.

La figure 1.2 représente un flot de conception de systèmes embarqués. Le système est initialement représenté par un ensemble de modules interconnectés au travers de ports par des canaux de communication. Cette spécification de haut niveau est désignée comme modèle « fonctionnel ».Ce modèle fonctionnel est raffiné dans un premier temps pour déterminer les choix architecturaux tels que le partitionnement logiciel/matériel des implémentations des fonctionnalités, les protocoles de communications et les diverses allocations d'unités de calcul. Pour cela, des regroupements hiérarchiques de modules sont opérés afin de pouvoir leur allouer des ressources de calculs, tandis qu'une opération similaire est réalisée sur les canaux de communication. Chaque module et chaque canal représente alors une vue abstraite de son implémentation, et possède un comportement qui peut être caché au concepteur (propriétés intellectuelles ou « Intellectual Property (IP) »). Les itérations de cette opération, couplées à une évaluation macroscopique des coûts et performances de la solution architecturale constituent l'exploration d'architecture. Lorsque les décisions architecturales sont arrêtées, le flot de conception se scinde en trois branches adressant l'implémentation de 1º les composantes logicielles, 2º les parties matérielles et 3º les communications recouvrant ces deux domaines. Ainsi les modèles logiciels vont successivement traverser les niveaux *architecture du Système d'Exploitation (SE)*, *architecture générique* pour arriver au niveau jeu d'instruction ou « Instruction Set Architecture (ISA) ». Leurs contreparties matérielles seront déclinées en modèles comportementaux, avant d'être manuellement ou automatiquement, grâce à la synthèse de haut-niveau ou « High-Level Synthesis (HLS) », raffinées en modèles précis au cycle horloge près ou « Register Transfert Level (RTL) ». Quant à la communication, sa topologie et ses fonctionnalités seront spécifiées indépendamment de son implémentation logicielle ou matérielle dans le modèle *Macro-Architectural*. Puis des modèles propices à leur implémentation sont mis en œuvre, il s'agit de modèles RTL pour les implémentations matérielles et ISA pour les implémentations logicielles.

Ces niveaux d'abstraction et leur utilisation dans le raffinement graduel des modèles de systèmes sont repris dans la figure 1.3.

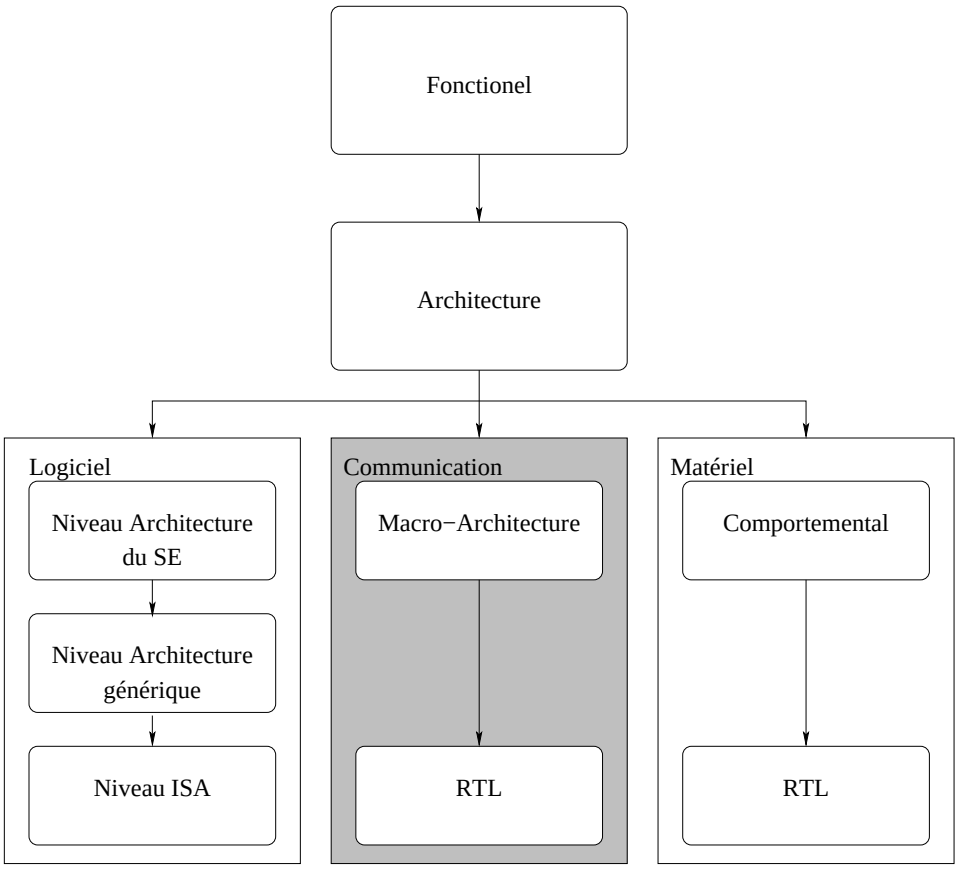


FIG. 1.3 – niveaux d'abstraction de la modélisation utilisés dans la conception.

1.3 Contributions

Les contributions de ces travaux trouvent place dans la modélisation de systèmes électroniques hétérogènes, l'accélération de l'étape d'implémentation de ces systèmes sur une architecture multiprocesseur hétérogène, une formalisation de l'interface de communication et leur génération par assemblage systématique. Il s'agit plus précisément de l'implémentation des communications entre les différents nœuds de calcul logiciels ou matériels. Cette action, localisée dans la branche grisée de la figure 1.3, a conduit à :

1. La définition d'un modèle de représentation d'architectures à composants hétérogènes.
2. La définition et l'implémentation d'un flot rapide et automatique de raffinement d'architectures multiprocesseurs monopuces spécifiques, en partant d'un modèle abstrait de l'application, pour obtenir par l'assemblage automatique et systématique de composants un modèle de réalisation. Ce raffinement comprend la génération des adaptateurs par combinaison systématique/spécialisation de composants génériques de bibliothèques.

Une dernière contribution adresse l'application de ce modèle de représentation et de ce flot de raffinement à quelques applications de complexité diverse, mais dont les plus significatives restent la partie émettrice d'un modem VDSL, la chaîne émission/réception d'un téléphone GSM WCDMA, un routeur de paquets.

1.4 Plan de ce mémoire

La suite de ce document est organisée en cinq chapitres. Le chapitre suivant présente une taxonomie des architectures et de leurs composants. Pour illustrer cette nomenclature des exemples sont ensuite étudiés. Puis un état de l'art des travaux relatifs au domaine adressé par cette thèse est proposé. Le chapitre 3 introduit les modèles de représentation d'architectures et d'application, tels qu'ils sont utilisés dans ces travaux. L'implémentation du flot développé, par un environnement d'aide à la conception assisté par ordinateur est détaillée dans le chapitre 4. La validité de cette approche est prouvée par son application à la conception de systèmes électroniques adressée par le chapitre 5. Le dernier chapitre est consacré à l'analyse rétrospective de ces travaux et à la suggestion de perspectives ouvertes par cette approche.

Chapitre 2

Architectures de systèmes multiprocesseurs monopuces

Sommaire

« A jeune cheval vieux cavalier. »

Proverbe Arabe.

Ce chapitre contient une étude des architectures matérielles multiprocesseurs. L'architecture matérielle d'un système est caractérisée par une structure et une organisation.

La **structure** est une représentation schématique et modulaire des composantes du système. Elle illustre aussi les interconnexions de ces modules et la direction des échanges d'informations entre ces derniers. La figure 2.1 représente un exemple de structure.

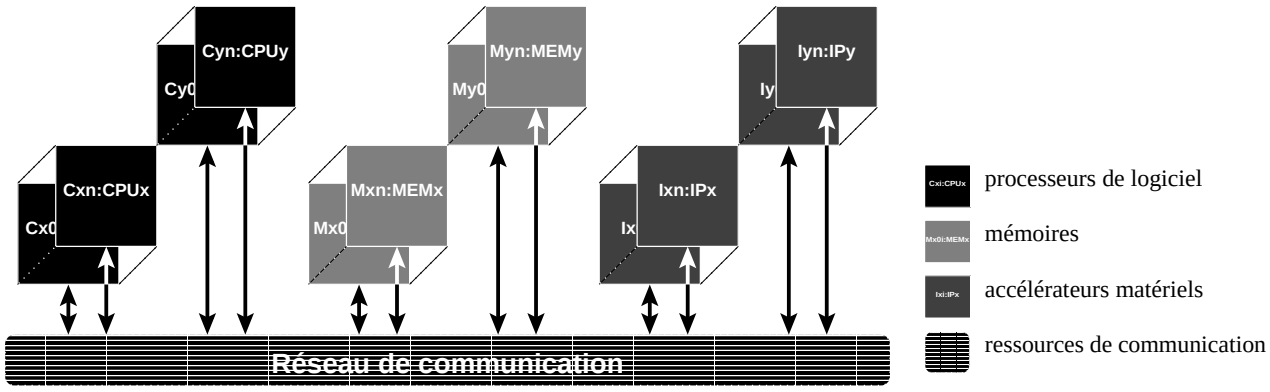


FIG. 2.1 – Représentation architecturale d'un système électronique monopuce.

Ce modèle est un assemblage de composants parmi lesquels on dénote :

– des instances de processeurs permettant d'exécuter du logiciel. Ces processeurs peuvent être de différentes natures (micro-controleurs, DSP, GPCPU, VLIW, etc...). on les désignera par l'expression « processeur logiciel » ;
– des composants de mémorisation pouvant de différents types (ROM, FLASH, SRAM, ...) ;
– des processeurs matériels, pouvant chacun être dédié à la réalisation d'une fonctionnalité précise ;
– un réseau d'interconnexion de tous ces composants leur permettant de communiquer.

L'**organisation** caractérise les relations de contrôle existantes entre les différents modules. Ainsi, un composant peut soit travailler en totale autonomie, soit opérer sous le contrôle d'un autre processeur, alors appelé maître. Ce composant (ou esclave) peut ainsi offrir l'exécution de services aux maîtres.

Le but de cette thèse est la génération automatique d'architecture à partir d'un modèle générique de référence. Ainsi, pour définir ce modèle, l'étude de quelques architectures est, ici, proposée. La première section de ce chapitre passe en revue quelques architectures et analyse leur structure et leur organisation. Cette étude ne prétend pas couvrir l'ensemble des modèles d'architectures existants. Nous ne nous intéresserons qu'à la régularité et à l'hétérogénéité d'architectures issues d'une opération plus ou moins systématique d'assemblage. La régularité concerne la manière dont les composants sont interconnectés (mise en évidence d'un motif architectural), alors que l'homogénéité caractérise la non-diversité des natures intrinsèques de chacun des composants.

Les trois sections suivantes aborderont l'architecture comme un modèle structurel. Les familles de composants constituant cette structure sont 1^o les composants de calcul, 2^o les composants de mémorisation, 3^o les composants de communications et, enfin, les composants d'entrée/sorties. Les trois premières seront introduites dans les sections 2.1, 2.2 et 2.3. Les membres de la quatrième famille fournissent des implémentations de services de communication fortement standardisées. Ils sont suffisamment définis pour que leur conception n'est pas à être adressée par ces travaux.

On distingue, dans la littérature classique[96, 36, 28], quatre grandes familles d'organisation des composants de traitement. Ce sont 1^o les architectures monoprocesseurs, 2^o les SIMD, 3^o les MISD, et finalement les MIMD. Le tableau 2.1 donne une classification de ces familles en fonction de leurs aptitudes à gérer plusieurs flux de données et/ou plusieurs flux d'instructions. Ces familles sont présentées dans les sections 2.4 et 2.5. Les organisations des accès aux mémoires par les différents composants de calcul seront introduites en section 2.6.

La caractérisation des architectures en terme de régularité sera apportée par la section 2.7.

Quelques architectures multiprocesseurs monopuces significatives sont étudiées en section 2.8 à fin de définir un modèle d'architecture ciblé par ces travaux. Avant de conclure en section 2.10, la section 2.9 présentera une modélisation générique de ces architectures hétérogènes ciblées afin d'en systématiser l'assemblage.

2.1 Les composants de calculs

Un composant de calcul est un composant matériel permettant l'implémentation plus ou moins flexible d'un algorithme de contrôle et/ou de traitement de données. On distingue essentiellement deux sous-familles de composants de calculs :

1. les processeurs matériels ;
2. les processeurs permettant l'exécution de logiciel.

Une présentation et une classification de ces composants sont proposées dans la suite de cette section. Les clefs de cette taxonomie sont : Ces composants de calcul peuvent être génériques ou dédiés à une fonctionnalité donnée. Leur utilisation permet l'exécution localisée d'une composante fonctionnelle. Leurs performances en terme de vitesse de traitement, leur coût en surface de silicium requise pour leur fabrication et leur consommation d'énergie sont des critères pouvant être considérés comme objectif de la conception ou au contraire comme contraintes. L'effort nécessaire à leur utilisation est devenu un critère supplémentaire car il influe directement sur la durée de conception du système. Les facilités de test pouvant être associé au composant prennent eux aussi une place de plus en plus prépondérante dans les décisions architecturales. Enfin la flexibilité inhérente à ces composants doit être étudiée. Celle-ci peut adresser les possibilités d'évolution des fonctionnalités du composant (ou flexibilité du comportement) mais aussi l'adaptation de ses communications à son environnement d'utilisation.

2.1.1 Processeurs matériels

Il s'agit de l'implémentation matérielle, encore appelée « logique cablée » d'un algorithme. Concrètement, un réseau de portes logiques (combinatoires et séquentielles) est utilisé pour implémenter de façon spécifique un comportement donné. Leur structure interne exhibe généralement la présence d'une partie contrôle composée d'une (rarement plusieurs) unité de contrôle réalisée par des machines d'états de Moore ou « Finite State Machine (FSM) », et une ou plusieurs unités de traitement constituant la partie opérative. Les termes « Application Specific Integrated Circuit (ASIC) », processeurs matériels, accélérateurs leur sont souvent consacrés. Ces processeurs sont utilisés lorsque :

1. l'algorithme à implémenter est définitivement arrêté (fonctionnalités, normes et standards fixés) ;
2. les performances requises (puissance de calcul, consommation, surface) pour l'exécution de cet algorithme ne peuvent être atteintes que par l'utilisation de ressources d'implémentation dédiées ;
3. le coût prohibitif de leur conception peut être amorti par un marché volumineux.

Ces implémentations spécifiques apportent des réalisations performantes et sobres tant en consommation énergétique qu'en occupation surfacique. Les performances en vitesse et la surface requise sont grossièrement estimables après la synthèse en portes logiques par des outils tels que [89] et plus finement par les outils de placement et routage tels que [13]. La consommation énergétique peut être estimée statiquement (par analyse) ou dynamiquement (par simulation) à partir de modèles en portes logiques ou transistors par [14], ou bien à partir de modèles plus abstrait (jeux d'instruction) [32].
La conception de tels composants est lourde en terme d'effort de développement, de validation de spécifications souvent très volumineuses. La testabilité de ces composants est atteinte par l'ajout de logiques additionnelles permettant l'injection de stimuli et l'extraction des réponses de façon sérielle (TAP+JTAG). Ces composants sont très peu flexibles. Leurs fonctionnalités ne peuvent être changées de façon significative, seule une restriction de leur usage selon une configuration donnée peut être opérée. Leurs interfaces sont elles aussi souvent spécifiques. Il est donc nécessaire d'adapter ces interfaces lorsque le processeur doit être connecté à un réseau de communication qui lui est incompatible.

Flux d'instructions	Flux de données	Nom	Exemples
1	1	SISD	Von-Neumann et Harvard
1	>1	SIMD	Machines vectorielles ou matricielles
>1	1	MISD	???
>1	>1	MIMD	Multiprocesseurs, multi-ordinateurs

TAB. 2.1 – Classification des organisations par Flynn[28].

2.1.2 Processeurs de logiciel

Il s'agit d'un cas particulier de processeurs matériels exécutant un algorithme d'interprétation d'un jeu d'instruction. Ces composants se décomposent en deux parties : 1^o la partie opérative elle-même composée d'une ou plusieurs unités fonctionnelles de complexité variable telles que les UAL et les MAC ; 2^o la partie contrôle en charge de séquencer les étapes de l'algorithme et de configurer la partie opérative. Les étapes principales de l'algorithme exécuté sont :

1. lecture d'une donnée de configuration appelée « instruction » et calcul à priori de l'emplacement de la prochaine instruction ;
2. décodage de l'instruction en une configuration de la partie opérative ;
3. lecture des données à traiter : les « opérandes » ;
4. attente de l'exécution des fonctionnalités par la partie opérative, avec potentiellement une correction de l'emplacement de la prochaine instruction ;
5. sauvegarde des résultats.

L'exécution de ces étapes est séquentielle car tant les ressources de calcul de la partie opérative que les ressources de contrôle de la partie contrôle sont limitées. Les processeurs de logiciels ne peuvent que rarement interpréter concurremment plus d'un algorithme applicatif.
Lorsque la vitesse d'exécution et la consommation énergétique requises ne nécessitent pas l'utilisation et la conception d'un processeur matériel spécifique, les processeurs de logiciel sont (ré)utilisés. Leur usage peut aussi être dicté par la nécessité de flexibilité. Il s'agit de pouvoir changer les fonctionnalités du système afin de suivre les évolutions de standards et de normes en cours de développement. Mais aussi de pouvoir réutiliser l'ensemble d'une architecture matérielle en reportant au sein de ses processeurs de logiciels les adaptations nécessaires pour supporter de nouvelles applications.
Les processeurs logiciels sont réputés lents ([109]) (au regard de leurs contreparties ASIC). En effet, à unités fonctionnelles équivalentes (natures et nombres), là où un processeur spécifique nécessitera un unique cycle d'horloge pour exécuter une opération, un processeur de logiciel peut nécessiter une demi-douzaine de ces cycles pour décoder et exécuter l'instruction équivalentes. Cependant cette analyse se restreint à l'exécution d'une unique instruction/opération. Pour un algorithme complet, il faut reconsidérer le problème en termes de blocs de plusieurs dizaines d'instructions/opérations. Or les processeurs de logiciel bénéficient d'un mécanisme appelé « pipeline » permettant de mettre en œuvre une concurrence locale de l'interprétation des instructions telle qu'elle est présentée précédemment. Le débit d'exécution de ces dernières peut alors atteindre la valeur limite d'une instruction par cycle d'horloge. Malheureusement, cette valeur ne peut être atteinte que localement, lorsque le « pipeline » peut complètement opérer (pas de rupture ou saut dans le flot d'instructions), et lorsque les données et instructions sont accessibles en des délais acceptables (cf. 2.6). De plus des limitations de la partie contrôle, de l'organisation de la partie opérative font que le parallélisme est restreint. Finalement les processeurs de logiciel sont évidemment plus lents que les processeurs matériels spécifiques, mais suivant la nature de l'algorithme (localité des données et des instructions) le rapport des vitesses d'exécution de ces deux implémentations fluctue entre 6 et l'unité. En pratique, la qualité des chaînes de développement (compilateurs, générateurs de code) permettent à ce rapport de rester proche de l'unité. La quantité de silicium requis par un processeur de logiciel n'est pas en relation directe avec la complexité ou le volume de l'algorithme applicatif à exécuter. Cette surface occupée peut donc être considérée comme constante, mais celles des mémoires en charge de contenir les instructions modélisant l'algorithme, ainsi que les données à traitées sont quant à elles, proportionnelles à la complexité algorithmique (cf. 2.2). La technologie Complementary Metal-Oxyd Semiconductor (CMOS) utilisée pour implémenter les systèmes électroniques consomme un quantum d'énergie à chaque changement d'états d'un transistor. La consommation énergétique est donc proportionnelle à l'activité des processeurs. Or un processeur de logiciel étant sensiblement lent, les concepteurs tendent à utiliser des fréquences d'horloges élevées, voire très élevées (qq. Giga-Hertz), augmentant ainsi considérablement la consommation. Pour limiter cette consommation, quelques processeurs de logiciel sont spécifiquement conçus pour les applications embarquées qui par définition ne disposent pas d'une source d'alimentation énergétique pérenne. Ils ont la faculté de pouvoir mettre « hors-tension » ou déconnecter de l'horloge, certains sous-ensembles de leurs unités fonctionnelles. Ce basculement en un mode faible consommation se fait explicitement par l'exécution d'une instruction particulière, alors que le retour est opéré sur détection d'événements. Cependant les délais nécessaires à ces basculements font que ces derniers ne sont utilisés que lorsque l'application n'est pas stressée par des contraintes temps réelles trop fortes. La pratique veut qu'un processeur de logiciel soit énergétiquement plus gourmand que son homologue spécifique.

Du fait des mécanismes utilisés pour augmenter les performances, la conception des processeurs de logiciel est si complexe et coûteuse qu'elle est laissée à la discrétion d'une communauté de spécialistes. Les travaux développés dans cette thèse se positionne plutôt autour de l'architecture système qui est utilisatrice des produits de cette communauté. Aussi les seuls efforts considérés sont ceux consentis lors de l'utilisation de processeurs de logiciels. Ils sont relatifs à l'apprentissage de la chaîne de développement relative, à la configuration et l'adjonction de quelques composants périphériques (tels que les mémoires) nécessaires à son fonctionnement. A ceux-ci s'ajoutent bien évidemment les efforts de modélisation de l'application qui, grâce à l'utilisation de langage de programmation de haut niveau, sont bien plus léger que lors d'une implémentation matérielle. Ces « coûts de conception » relativement faibles constituent le premier attrait des processeurs de logiciel.

Les concepteurs de ces composants étant conscients de leurs utilisations dans des applications diverses et variées, les ont munis d'une panoplie d'outils et instrumentations de test offrant :

– des simulateurs fonctionnels ou Instruction Set Simulator (ISS) modélisant l'exécution des instructions.
– des simulateurs modélisant au cycle horloge près les différentes étapes de l'algorithme de traduction des instructions. De tels simulateurs peuvent aller jusqu'à fournir une évaluation de la consommation énergétique. De plus, il est possible d'adjoindre à de tels simulateurs des modèles fonctionnels du bus du processeur ou « Bus Functional Model (BFM) » capables de simuler les signaux de ce dernier.
– des émulateurs constitués d'un environnement logiciel et d'un circuit imprimé regroupant des composants discrets. Les différentes unités du processeur sont implémentées par des composants distincts dont les inter-connections peuvent être espionner afin de mieux connaître l'état interne du processeur.
– des « test-chip » ou circuits tests. Ce sont des versions du processeur auxquelles ont été ajouté des unités d'espionnage (TAP et *boundary cells* du standard IEEE 1149 JTAG) permettant l'extraction des variables d'états du plus profond du processeur jusqu'à son extérieur, tout en utilisant un nombre réduit d'entrées/sorties additionnelles.

Il est donc relativement aisé de tester les différents modèles des sous-ensembles du système destinés à une implémentation par processeur logiciel. Les rares difficultés rencontrées sont relatives aux liaisons entre le modèle de validation et les modèles du reste du système (environnement). Ces validations sont utilisées très tôt dans le flot de conception en utilisant les ISS, mais aussi après fabrication pour une mise au point plus poussée de l'implémentation ou tout simplement pour vérifier son état de marche. Les processeurs de logiciel sont les champions de la flexibilité fonctionnelle : tout algorithme pouvant être exprimé par un programme ou ensemble d'instructions peut être implémenté par un processeur de logiciel. En pratique tous les jeux d'instructions forment un langage complet[1]. Moyennant quelques concessions en performances, il est toujours possible (en admettant le respect des contraintes temporelles et de consommations énergétique et surfacique) d'implémenter un algorithme numérique par un processeur de logiciel. Enfin, les processeurs de logiciel sont conçus avec une interface compatible avec un unique médium de communication, communément appelé « bus natif ». Ils opèrent en maître sur ce bus et donc sur tous les autres composants connectés (périphériques). L'ensemble processeur de logiciel + bus + mémoires + périphériques forment une Architecture Locale (AL). Il est possible d'associer plusieurs processeurs pour gagner en parallélisme de deux façons :

1. en connectant plusieurs AL pour obtenir une « architecture multi-ordinateur » ;
2. en connectant plusieurs processeurs de logiciel autour d'un unique réseau de communication, formant ainsi une « architecture multiprocesseur » (cf. 2.7.1).

Dans ces deux schémas d'extension, lorsqu'une incompatibilité sur les moyens de communication intervient, soit entre deux AL, soit entre un processeur et le réseau de communication, une unité d'adaptation doit être insérée. Contrairement aux processeurs matériels, cette unité peut partiellement être réalisée en logiciel réduisant ainsi leurs coûts et augmentant sa flexibilité. L'algorithme exécuté sur un processeur peut être déterminé par :

– une configuration statique ou programmation consistant en une écriture des instructions à exécuter dans les mémoires reprogrammables associées au processeur. Cette opération est lourde et peut nécessiter du matériel externe. Elle est donc extraordinaire, voire unique.
– une configuration explicite ou chargement de programmes par recopie de blocs d'instructions d'un emplacement mémoire (ou périphérique tels que des unités de communication) vers un autre. Ces reconfigurations prennent place lors d'appels explicites dictés par les actions de l'utilisateur final de l'application.

[1] Un langage est dit complet s'il permet de modéliser la machine de Türing[101].

– une reconfiguration implicite et fortement dynamique opérée par recopies de blocs d'instructions d'emplacement mémoire vers un autre. Ces opérations sont très régulièrement déclenchées par le programme exécuté. On peut ainsi mettre en œuvre un environnement multitâches simulant l'exécution parallèle de plusieurs branches concurrentes d'un algorithme par l'exécution alternée de petites tranches de ces branches. On parle alors de « changement de contexte », le contexte étant un enregistrement de l'état courant d'une branche.

2.2 Les composants de mémorisation

Les composants de mémorisation ou mémoires sont des ressources matérielles permettant de stocker pour des durées et des usages variables, les données à traiter, les résultats temporaires, ainsi que des informations de configuration telles que instructions pour les processeurs de logiciel.
Les adjectifs « vive » et « morte » permettent de qualifier la durée de vie du contenu d'une mémoire. Une mémoire « vive » est une mémoire dont le contenu peut être modifié, alors qu'une mémoire « morte » est figée. La mémoire « morte » peut conserver durablement ces données alors que sa contrepartie « vive » perdra son contenu à chaque mise hors tension de l'application. De par leur nature les mémoires de ces deux familles ne sont pas destinées à la même utilisation. Une mémoire « morte » ou *Read-Only Memory (ROM)* retiendra des informations immuables telles que des instructions ou des données constantes pour un processeur logiciel. Les mémoires « vives » ou *Random Access Memory (RAM)* sont utilisées pour conserver les informations variables telles que des résultats temporaires de calculs, mais aussi les instructions de programmes qui ne sont pas exécutés de façon systématique et récurrente par le processeur.
Afin de permettre l'évolution des applications, des mémoires « mortes » dont le contenu peut être effacé puis réécrit ont été conçues, les *EPROM* pouvant être effacées par Ultra-violets ou électriquement : les *EEPROM*. La dernière évolution de ces mémoires « mortes » évolutives sont la mémoire *FLASH* et la mémoire *FRAM,* dont les hautes intégrations permettent un usage massif. La pérennité de données entre deux utilisations de l'application séparées par une mise hors tension, indispensable pour reprendre le traitement de l'information là où il s'est interrompu à la dernière mise hors tension, nécessite l'utilisation de *RAM* sauvegardée (c-à-d. possédant sa propre alimentation électrique) ou bien d'*EEPROM* ou *FLASH* lorsque les temps d'accès en écriture ne sont pas critiques.
La famille des mémoires peut aussi être décomposée selon les modes d'accès supportés. Les accès « séquentiels » sont caractérisés par la connaissance, à tout moment de l'emplacement du prochain accès : *First-In First-Out memory (FIFO), Last In First Out memory (LIFO)*. Quant aux accès « aléatoires », la mémoire ne peut pas anticiper l'emplacement du prochain accès et doit donc en être informé par une information supplémentaire : l'adresse. Une décomposition supplémentaire de ces deux sous familles permet de caractériser la réponse d'une mémoire à un accès. Si la mémoire répond aux fronts du signal d'horloge la séquençant, on parle alors d'accès *synchrone* et de *Synchronous Dynamic Random Access Memory (SDRAM)*. Sinon, si la mémoire répond au terme d'un délai fixe ou variable (indiquant alors la completion de l'accès à son instigateur), elle sera désignée par le terme *asynchrone*.
Les mémoires sont utilisées dans différents cas : 1^o fournir un environnement constant pour assurer un comportement reproductible de chaque exécution du système ; 2^o transmettre des informations d'une session de fonctionnement à la suivante ; 3^o répartir temporellement et spatialement l'utilisation de ressources afin d'en diminuer le nombre grâce à :
– la sauvegarde de résultats temporaires pouvant être réinjectés dans des itérations futures du même calcul (boucles), ou dans d'autres branches de l'algorithme dont l'exécution est prévue plus tard pour cause de dépendances de données et/ou disponibilités de ressources.
– le découplage des localisations des données et des unités de calculs, permettant une assignation dynamique de ressources pour le traitement des données (cf. 2.7.1 architecture symétrique et répartition de charge).

Il est possible de rencontrer au sein d'une architecture monopuce, les technologies d'éléments de mémorisation suivantes :
– Des unités de mémoire de masse, telles que des blocs FLASH ou FRAM, contenant des données volumineuses et faiblement volatiles ;
– Des unités de mémoire morte, de faibles capacités, contenant les instructions de programmes dédiés aux premières étapes de la mise en fonction de l'application, ainsi qu'à sa mise à jour par reprogrammation de mémoires FLASH.
– Des blocs de mémoires vives *Dynamic Random Access Memory (DRAM)* ou *Synchronous Dynamic Random Access Memory (SDRAM)* pour les données volatiles.
– Des petits blocs *Cache*s composés de mémoires SRAM de faibles capacités et de contrôleurs matériels en charge de maintenir une copie des instructions et/ou données régulièrement accédées ou voisines des derniers accès. Cette

technique permet de bénéficier de la rapidité des SRAM, sans être pénalisé par leur encombrement.
– La dernière couche mémoire est enfouie au plus profond des processeurs. Il s'agit des bancs de registres, ou chaque registre est un regroupement de bascules bistables, dont l'accès à chaque unité de taille atomique (ou *mot*) est rendu direct par leurs connexions explicites.

Les performances en vitesses des mémoires se résument à la bande-passante (ou débit d'information) qu'elles peuvent supporter. Cette métrique dépend de trois facteurs :

1. le temps d'accès en lecture/écriture. Ces temps sont inhérents à la technologie et la structure des points de mémorisation. De plus, à cause des capacités parasites intrinsèques des entrées/sorties des points mémoires ou *binary digit (bit)*, ces temps d'accès sont fortement corrélés au nombre de ces derniers. Puisque les horloges des composants de calcul et de communication sont de plus en plus rapide, Il est de plus en plus courant d'imposer à ceux-ci des cycles d'attentes afin que les mémoires puissent répondre aux requêtes qui leur sont adressées.
2. la largeur du chemin de données.
3. des mécanismes internes (rafraichissement, pagination) pouvant la rendre indisponible durant quelques cycles d'horloges.

Pour les mémoires comme pour tous les autres composants CMOS, l'activité (ou basculement) des transistors est consommatrice d'énergie. Ceux-ci sont actifs lors des accès et des éventuelles session de rafraîchissement. La consommation est principalement due à ces accès, bien que pour les technologies sub-microniques actuelles, les courants de fuites dans le substrat engendre un surcoût énergétique de plus en plus important. La surface de silicium nécessitée par les mémoires représente le plus grand taux d'occupation surfacique pour les applications monopuces. Elle est estimée par [44] à 80% de la surface totale.
Du fait de leur haute intégration, des différents mécanismes mis en jeu pour améliorer leurs performances (raffraichissement, pagination), la complexité de leur conception en font l'apanage d'équipes hautement spécialisées. Dans les présents travaux, nous ne nous positionneront qu'en tant que réutilisateur de blocs mémoires et/ou d'outils de génération de leurs modèles. Les mémoires utilisent des technologies diverses dont certaines sont extrêmement sensibles aux radiations, lumières et bruit électro-magnétique et qui plus est, cohabitent péniblement sur une même puce. Aussi certaines d'entre-elles embarquent de la logique supplémentaire telle que le Built-In Self Test (BIST) pour tester leur comportement et éventuellement décider de « remplacer » les parties endommagées par des éléments de secours (« redondance logique »).
L'organisation des bits au sein d'une mémoire répond à une règle inflexible : Une mémoire est organisée en $n = 2^p$ mots de $w = 2^r$ bits, avec $\{n, p, w, r\} \in \aleph^4$. Leur interface est conçue pour supporter des protocoles et modes de transferts donnés. Il peut donc se révéler nécessaire de leur adjoindre des adaptateurs pour les connecter au réseau de communication. Enfin, leur comportement passif et esclave peut être enrichi par l'association de contrôleurs additifs tels que des DMA et contrôleur de FIFO

2.3 Les composants de communication

Les liens reliant les nœuds de calcul ou de mémorisation ont eux-aussi une variété de concrétisations physiques.
Ces composants permettent l'implémentation des liens logiques de communication reliant les nœuds de calcul ou de mémorisation. Ils sont composés de ressources de routage et de transport de l'information, mais aussi d'unité de décision de l'attribution de celles-ci lorsque leur usage est partagée par l'implémentation de plusieurs liens logiques concurrents. Leurs utilisations peuvent être :
– l'échange de données entre interlocuteurs (composants de calcul et de mémorisation). On associe les termes chemin de données ou « data-path » à cette utilisation performante, mais très déterministe des composants de communication ;
– la synchronisation et la configuration/contrôle entre deux composants de calcul distants et concurrents. On parle alors d'utilisation non-déterministe et fortement orientée contrôle de l'application.

Les performances d'échange de l'information dépendent de la topologie de ces composants et de la bande passante de chacun des sous-composants impliqués dans la communication, du nombre de liens logiques se partageant les mêmes ressources, mais aussi du protocole d'échange. La consommation des composants de communication est soigneusement réduite car ce sont de puissants émetteurs et de sensibles récepteurs de bruits électro-magnétiques, pouvant se parasiter les uns et les autres. L'occupation surfacique de ces composants doit rester faible face à l'occupation de la logique

cablée implémentant les unités de calcul. Il faut donc prendre garde à ce que le choix d'une structure de ces composants ne vienne pas renverser cette relation d'ordre.
Les fonctionnalité de ces composants sont très basiques, mais leur dimensionnement qui influe directement sur les performances de l'ensemble de l'application doit être soigné. Les plus complexes de ces composants peuvent embarquer de la logique de tests permettant d'espionner leur états internes. Pour les plus simples, le matériel de test des composants de calcul ou de mémorisation connectés aux terminaisons suffit amplement à vérifier la bonne transmission des informations. La flexibilité d'un composant de communication peut être considérée sous trois angles :

1. l'extensibilité ou l'aptitude du composant a supporter un nombre donné d'utilisateurs concurrents sans altérer ni ses fonctionnalités, ni ses performances globales. Les anglo-saxons consacrent le terme de « scalability » à cette propriété, nous lui préférerons l'expression « mise à l'échelle ».
2. la compatibilité aux schémas de communication utilisés par l'application : point-à-point, multipoint, transferts en rafale, transferts non-préemptifs[2].
3. la compatibilité avec d'autres réseaux éventuellement de natures différentes.

Ces trois points font que des modules matériels doivent être insérés pour les connexions inter-réseaux et réseau-nœud de calcul/mémorisation. Ce sont, respectivement, les ponts et les adaptateurs de communication.

2.3.1 Bus partagé

Il s'agit d'implémenter plusieurs liens logiques de communication par un unique médium de communication à accès partagés : le *bus*. Ce médium permet la mise en place de liens point à point ou multipoints entre un maître[3] et un ou plusieurs esclaves.
Le bus est alloué dynamiquement à une communication, plus précisément à un maître, en fonction d'une police d'attribution gérée par l'arbitre du bus. En pratique, on distingue plusieurs polices d'attribution telles que :
– l'élection d'un maître parmi tous ceux qui réclament l'accès en fonction de priorités qui leur sont statiquement assignées ;
– le choix d'un maître parmi tous ceux qui sont en attente en fonction de priorités dont les valeurs sont dynamiquement et circulairement assignées. On parle alors d'ordonnancement par priorité tournante ou *tourniquet* ;
– l'attribution systématique et cyclique par tranche de temps ou *Time Division Multiple Access (TDMA)* du bus à tous les maîtres (comprenant ceux qui n'en n'ont pas l'usage) ;
– l'élection systématique et cyclique d'un maître par libération et circulation d'un jeton (*Token-Ring*), sans tenir compte de la réelle nécessité ;
– l'élection par priorités dynamiques en fonction de la périodicité des communications (Rate Monotonic Analysis (RMA)) ;
– l'attribution par priorités dynamiques selon l'ordre d'apparition des requêtes ou *First-In First-Granted (FIFG)* ;
– l'élection par priorités dynamiques en fonction de la proximité d'une communication de son échéance ou *Earliest-Deadline First (EDF)*.
Les échanges peuvent être cadencés par :
– Une horloge globale (système) : bus globalement synchrone ;
– Une horloge spécifique au bus : bus localement synchrone et globalement asynchrone ;
– Par l'état du maître et des esclaves : bus asynchrone.
Les deux dernières catégories permettent de faire communiquer des éléments de l'architecture ayant des horloges indépendantes, alors que la première nécessite l'utilisation de la même horloge pour tous les composants connectés au bus.
La structure de tels composants peut encore se décomposer en une partie contrôle et une partie chemin de donnée. Le chemin de donnée est constitué de ressources de transport généralement implémentées par des pistes métalliques ;
Plusieurs fonctionnalités incombent à la partie contrôle des composants de communication :

1. l'attribution du chemin de donnée ;
2. la synchronisation des extrémités ;
3. le routage de l'information.

[2] nécessaire pour les synchronisations par sémaphores
[3] maître : unité initiant la communication.

Ces bus sont généralement utilisés pour des systèmes ne nécessitant pas de larges bande-passantes entre plusieurs composants de calculs. Ils sont donc employés dans les applications orientée contrôle, pour la configuration de composants de calcul (*data-path*), mais aussi comme bus local dans les AL.
Les bus Advanced Multi-masters Bus Architecture (AMBA)[1], Integrated Peripherals Bus (IPBus)[42], Peripheral to Central Processing Unit Interconnects (PCI)[72] sont des exemples de bus partagés. Ils peuvent supporter la connexion de plusieurs composants initiateurs de communication (Multi-Master), et doivent donc gérer en temps réel l'attribution des médium pour une communication logique entre deux interlocuteurs. Les vitesses de fonctionnement de ces bus en font des unités de communication très performantes. Cependant, cette caractéristique doit être revue à la baisse en fonction de l'activité et du nombre des communications se partageant cette unité. En effet, plus un bus est partagé plus la latence moyenne d'attribution est élevée. Lorsqu'ils ne sont pas disponibles pour une communication, on les dits « bloquants ». Ils gèlent ainsi l'exécution de parties de l'application. En tant que goulot d'étranglement des performances globales, leur propre performance devient donc un critère limitant et doit en tant que telle, être soignée. Du fait du nombre restreint de ressources, les bus partagés consomment peu d'énergie. Cependant ils peuvent pousser les composants qu'ils connectent à consommer plus. En effet, le partage de ressources implique inévitablement des cycles d'attente d'attribution durant lesquels de l'énergie est inutilement gaspillée. L'encombrement de cette solution est la plus réduite. Le partage intensif d'un nombre restreint de ressources offre le meilleur rapport coût/nombre de communications. La conception de ces composants doit sa complexité à la recherche de meilleures performances. Leur réutilisation qui réellement concerne l'architecte du système, se limite à configurer statiquement ces composants pour obtenir la bande passante et l'ordonnancement des communications adéquats. Le test de cette famille de composant est indirectement réalisé par le test effectif des composants connectés en périphérie. Ces composants de communication sont :

1. non-extensible, une augmentation du nombre de composants de calcul connectés réduit drastiquement les performances globales des communications.
2. compatible uniquement avec une famille ou un jeu restreint de protocole.

Le 2^o point se résout par l'insertion d'adaptateurs de communication entre les composants de calcul/mémorisation et le bus. L'utilisation de plusieurs bus potentiellement de natures différentes permet de résoudre le 1^o point en fournissant localement les performances requises. Mais cette solution n'est réalisable qu'en découplant les différentes instances de bus par des ponts.

2.3.2 Les réseaux de communication

Les réseaux de communication se différencient des bus partagés par :
– leur nombre de ressources de transport et de routage supérieur à l'unité ;
– la distribution du routage sur l'ensemble du réseau ;
– la possibilité d'intégrer des services.
Ils prennent place dans la réalisation d'applications « temps-réel » et « traitement du signal » nécessitant une grande bande-passante, et une faible latence. Celles-ci mettant en œuvre des calculs massivement parallèles, leurs nombreuses communications à haut-débit ne peuvent être réalisées que par un réseau.
L'utilisation d'un nombre élevé de ressources de transport et de routage permet d'offrir de manière quasi-permanente la disponibilité des unités de transports. Les solutions à base de réseaux de communication offrent donc les meilleurs débits et les latences les plus basses. Le nombre de ressources utilisées augmente en proportion la consommation du réseau. De plus, les unités de contrôle et de routage essaimées sur toute la surface du réseau consomment inutilement lorsqu'ils ne sont pas sollicités. L'augmentation des ressources se traduit bien évidemment par un accroissement de l'occupation surfacique. Mais si la surface nécessitée par les unités de contrôle et de routage est proportionnelle à leur nombre, il en va différemment pour les ressources de transport. En effet ces dernières utilisent des pistes métalliques, dont on dénombre jusqu'à huit niveaux sur les puces modernes. En pratique, la logique cablée ne nécessite pas plus de quatre de ces niveaux, laissant ainsi les deux niveaux supérieurs disponibles pour l'implémentation des communications. Mais les réseaux de communication fortement parallèles peuvent nécessiter tant de pistes que ces deux niveaux ne suffisent à les fournir. Il faut alors définir sur la puce, des emplacements réservés au réseau de communication afin de bénéficier de la totalité des niveaux métalliques. Du fait de la complexité de leur structure et de leurs algorithmes de routage, les réseaux de communication sont de véritables challenges relevés par des concepteurs spécialisés. Leurs caractères génériques et configurables nécessaire pour leur support d'un nombre variable de

connexions sont pour l'essentiel des causes de cette complexité. Cependant, leur réutilisation est fortement aisée par la philosophie de généricité qui a guidé leur conception. Le test du bon transport de l'information par unique observation de la périphérie du réseau ne suffit plus. Les nombres élevés d'unités de routage, de chemins possibles pour une même communication poussent à l'intégration de fonctionnalité de test au sein du réseau. Le nombre de connexions supportées par ces réseaux est rendu facilement extensible par leur régularité structurelle. Quand aux protocoles des composants de calcul connectés aux extrémités du réseaux, leur incompatibilité peut ici encore, nécessiter l'usage d'adaptateurs de communication. Parmi les différents réseaux de communication pour systèmes sur puce, nous étudierons 1^o les réseaux Cross-Bar, 2^o les réseaux commutés et 3^o les bus micro-commutés.

Réseau Cross-Bar

Il s'agit de la mise en parallèle de n lignes de transmission concurrentes dont l'attribution est régie par une police TDMA. La figure 2.2 en dépeint la structure.

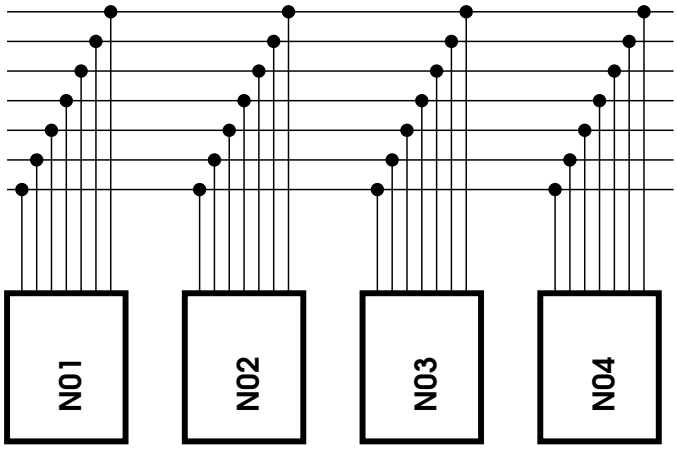


FIG. 2.2 – Bus Cross-Bar.

Le réseau de processeur s'articule autour d'une matrice (appelée « FPIC » ou « Switch Fabric ») d'interrupteurs programmables permettant la mise en œuvre concurrente de plusieurs canaux de communication.
La configuration de cette matrice peut être :
- Statique par programmation de la matrice lors de la fabrication, élevant ainsi la réutilisabilité de la plateforme ;
- Faiblement dynamique par reconfiguration des interconnexions à chaque démarrage de l'application ;
- Fortement dynamique lorsqu'au cours de l'exécution de l'application, une reprogrammation est décidée pour mieux répondre aux besoins courants en communication.

Lorsque ce réseau forme un graphe fortement connexe[4], il est dit « full-connect ». Les implémentations de Cross-Bar sont généralement synchrones. On retrouve ces bus lorsqu'une grande bande-passante et surtout une latence faible sont nécessaires. Il s'agit des applications à composantes fortement orientées calcul. Ils sont aussi utilisés comme bus de cohérence pour le mécanisme de cache système dans les architectures multiprocesseur.
Sur de tels bus,le parallélisme des communications est élevé, ils sont très faiblement blocants. La vitesse de chacune des lignes d'un cross-bar est très élevée. L'ordonnancement TDMA et le nombre élevé de ressources rendent cette solution énergétiquement coûteuse. Le coût surfacique qui est proportionnel à n^2, rend difficile l'utilisation de ces bus pour les applications dites massivement parallèles. La conception de tels bus est assujettie de contraintes fortes concernant le placement sur la puce des lignes de transmission. Heureusement, la régularité de ces composants permet la mise au point de générateurs de modèles placés et routés sur silicium. La matrice n'opérant pas aucune décision sur le routage, le test est reporté aux interfaces des lignes de transmission. La disponibilité de modèles hautement paramétrés ou de générateurs dédiés permettent d'étendre aisément le nombre de lignes de transmission. L'usage d'adaptateur de communication n'est pas nécessaire, car il est aisé de spécialiser une ligne de transmission pour le protocole de ses extrémités. Évidemment, si les deux extrémités d'une ligne ne supportent pas le même protocole, alors l'une d'elle devra être adaptée.

[4]toute paire de sommets (nœuds de calcul) est reliée par un arc (ligne de transmission).

Réseaux commutés

Ce type de réseau comporte plusieurs étages, dont le franchissement s'opère au travers d'un commutateur en charge de router le message vers un canal parmi n. Pour des raisons de performances et/ou de tolérances aux pannes, il est possible d'introduire des chemins redondants permettant ainsi d'augmenter la concurrence des communications et les solutions de routage. Leur utilisation au sein d'architectures monopuces est abordée par [33].
L'implémentation de chaque commutateur peut être à base de multiplexeurs voire à base de petits bus Cross-Bar. La propagation de l'information peut suivre l'un des deux schémas suivants :

1. une connexion source-destination (aussi appelée « session ») permettant le transport continue et transparent de l'information au travers des étages ;
2. une propagation de l'information d'étage en étage où elle est temporairement bufferisée dans une unité de routage. Ce mode de transmission est appelé « worm-hole ».

Une configuration particulière de ces réseaux, le commutateur « Batcher-Banyan » est utilisée pour l'implémentation des routeurs de protocole Asynchronous Transfert Mode (ATM)[48] présentés par la figure 2.3.

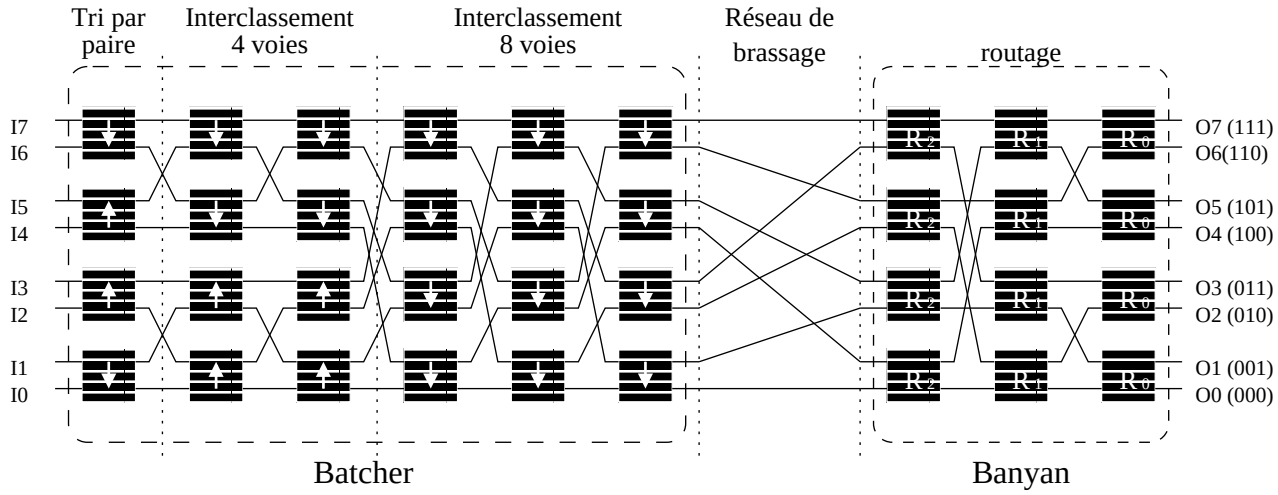


FIG. 2.3 – Réseau commuté ATM.

L'information s'y propage sous forme de paquets appelés cellules dont l'entête contient l'adresse de destination. Les routeurs du sous-commutateur « Batcher » sont des instances d'une de ces deux variantes :

Si l'adresse de la cellule entrante en I_0 est inférieure à l'adresse de la cellule entrante en I_1, elle sera routée vers O_0, tandis que l'autre cellule sera dirigée vers O_1. Si tel n'est pas le cas, alors c'est le routage complémentaire qui est appliqué ;

Pour ce routeur, la police de routage est renversée. Ainsi la cellule de I_0 est dirigée vers O_1 si son adresse est la plus faible et vers O_0 sinon. Quand à la cellule de I_1, elle suit le routage complémentaire.
Pour le sous-commutateur « Banyan » seuls des routeurs  sont utilisés. Leur police de routage est basée sur la valeur du $i^{\text{ème}}$ bit de l'adresse de la cellule : si cette valeur est nulle alors la cellule (entrée en I_0 ou I_1) est dirigée vers O_0, sinon vers O_1. Si les deux cellules entrantes doivent être routées vers la même sortie, on parle alors de collision, et une cellule doit être élue et transmise tandis que l'autre est bufferisée pour ne pas être perdue.
Les routeurs de « Batcher » permettent donc d'ordonner les cellules suivant la valeur de leur adresse alors que les « Banyan » décodent ces mêmes adresses pour router les cellules jusqu'aux destinations. L'association de ces deux technologies en une architecture « Batcher-Banyan » permet grâce au tri des cellules réalisé par le sous-commutateur « Batcher » une meilleur répartition géographique du trafic, diminuant ainsi les collisions et autorisant l'utilisation de buffers de tailles réduites au sein des routeurs « Banyan ».
La bande passante typique d'un bus commuté est proche de celle des bus cross-bar. Par contre, les latences qui sont proportionnelles à la profondeur ou nombre de commutateurs traversés (surtout en transmission « worm-hole ») peuvent être bien plus élevées. L'activité des routeurs est élevée afin d'atteindre une vitesse globale de transmission conséquente. Ces unités étant nombreuses, la consommation de l'ensemble du réseau est importante. Bien qu'étant

moins performants que les bus Cross-Bar, ces bus commutés leur sont préférés afin de réduire le coût de la mise en œuvre des communications, car leur complexité en $n\ log_2\ n$ (avec n nombre d'entrées) est bien moindre. Là encore, la réutilisation de composants placés, routés est nécessaire. L'usage de générateur de modèles largement éprouvés est préconisé. Chaque commutateur du réseau peut se voir associer de la logique de test autorisant ainsi le test en ligne et la modification dynamique de l'algorithme global de routage lorsqu'un chemin est défaillant. Ces composants peuvent être mis à l'échelle à condition de préserver leurs caractéristiques de régularité. Ainsi leur nombre de lignes peut se voir évoluer suivant une règle précise (ex : puissance de deux pour les réseaux ATM et Fat-Tree[33]).

Bus micro-commutés

Le bus μNetwork de Sonics[86] est une bonne illustration de cette technologie. La figure 2.4 en illustre la structure.

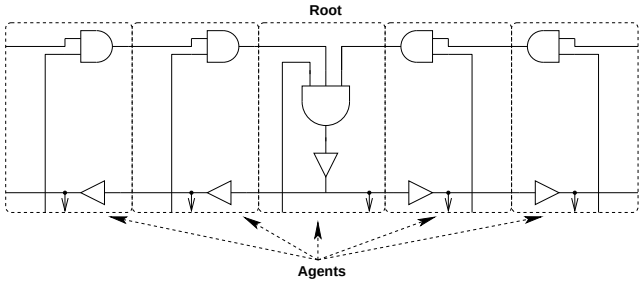


FIG. 2.4 – Couche physique du μNetwork.

Les éléments de commutation sont enfouis au sein même du bus, comme des macrocellules (rectangles pointillés). C'est l'assemblage de ces macrocellules qui constitue le médium physique de communication. Cette famille de bus est de plus en plus utilisée (avec une complexité variable) dans les systèmes monopuces. Ce sont même ces systèmes qui sont à l'origine de la création de cette famille de composants de communication.
Tout comme leurs ainés les bus partagés, ces bus ne possèdent qu'une unique route. Cependant, ils n'en sont pas pour autant bloquant. En effet l'attribution de l'accès au médium est régis par une règle équitable à base de police tourniquet ou « round robin » et attribution par tranches de temps ou « Time Division Multiple Access (TDMA) ». Ceci permet d'obtenir des latences acceptables et de garantir une bande passante. Ces caractéristiques sont dans la littérature regroupés sous la dénomination de « Qualité de Service ». La simplicité des macrocellules (quelques portes logiques chacune) permet de limiter la consommation qui se situe entre celle des bus partagés et celle des bus commutés. La simplicité des macrocellules a aussi pour conséquence de ne nécessiter qu'une surface réduite. Cependant, ces dernières empêchent l'implémentation du bus dans les couches métalliques supérieures et des zones du circuits devront lui être spécifiquement dédiées. Ces bus étant extrêmement régulier, leur conception est aisée. Le faible coût d'implémentation de ces bus étant un argument principal de leur utilisation, aucun matériel de test n'est adjoint aux macrocellules. Le test de l'ensemble du bus est reporté dans le test des composants en périphérie. L'extension des modèles de tels bus pour supporter un plus grand nombre de connexion peut se faire aisément grâce à leur régularité. De plus, la dispersion des éléments de routage (macrocellules) au sein-même du bus fait que celui-ci dispose de répétiteurs de l'information augmentant ainsi la distance critique séparant deux terminaisons. Cependant, on prendra garde à l'évolution de la fréquence d'horloge séquençant le bus qui, pour supporter les bandes passantes BW_i requises par les communications, doit suivre la loi : $f \geq \frac{\sum BW_i}{W}$, avec W la largeur des données véhiculées par le bus.

2.3.3 Les connexions point à point

Il s'agit de l'utilisation de ressources de transmission spécifiques et dédiées à l'implémentation d'un lien entre deux nœuds de calcul.
Le principal usage de ce schémas de communication prend place dans l'implémentation des communications entre un processeur et un accélérateur. Plus particulièrement lorsque l'application est orientée traitement de données et nécessite des latences faibles et des débits importants qu'une ressource partagée ne peut offrir. Les architectures cascadée (« pipeline ») en sont très friandes.
La ressource de transmission étant dédiée à une unique communication, elle est toujours disponible pour cette dernière et peut ainsi lui offrir sa pleine bande-passante. De plus, du fait de la simplicité du contrôle de telles connexions, les

débits y sont élevés et les latences quasi-inexistantes.
Du fait de leur usage pour l'implémentation d'une unique communication, la surface de ces connexions est proportionnelle à leur nombre.
L'effort d'intégration est très faible grâce à la complexité réduite des unités de contrôle. La testabilité est inexistante, et reportée aux tests des extrémités afin de ne pas apporter de surcoût prohibitif. Ces connexions sont non-extensibles et spécifiques à un protocole.

2.3.4 Les adaptateurs

Ce sont des blocs souvent matériels, rarement à base de processeur logiciel ([12] étant la seule exception connue). Leur utilisation est requise lorsqu'il est nécessaire d'adapter les couches « liaison de données » et « physique » du protocole natif d'une unité matérielle (processeur matériel, processeur de logiciel ou mémoire) aux mêmes couches du réseau de communication le reliant à son environnement. Les performances de ces composants sont fortement dictées par la nature de l'application et du réseau de communication :
– Pour les applications dominées par le contrôle, le non-déterminisme des communications rend difficile leur modélisation et contraint les outils d'exploration d'architecture à les négliger. La monopolisation des ressources de calcul pour traiter des tâches de communication nuit fortement aux performances de l'exécution des algorithmes applicatifs ;
– Pour les applications à dominantes calcul, bien que le déterminisme des communications et les mécanismes de « bypass » permettent de masquer les latences, le débit des communications doit cependant être suffisemment élevé pour suivre les cadences imposés par le cadencement des unités de calcul.
– Les ressources des réseaux de communication sont généralement partagées par un nombre massif de connexions logiques, l'utilisation trop longue de celles-ci peut empêcher une tâche de respecter les échéances qui lui étaient imposées.

Les fonctionnalités simplistes prises en charge par ces blocs ne nécessitent que très peu d'énergie. Les primitives de traduction mises en jeu ne réclament que peu de silicium. Il est courant d'estimer leur occupation surfacique à moins d'un dixième de la surface totale (sauf pour [12], mais ici le contexte de système monopuce est dépassé). Suivant la nature des deux protocoles, la nécessité de découpler le nœud de calcul du réseau par « bufferisation » et/ou l'usage de multiples horloges, la conception de ces unités fluctue entre la simple réutilisation de composants prédéfinis à la conception de blocs spécifiques. Les protocoles usités étant généralement issus de standards, cette étape est limitée à l'instanciation de composants et à leur dimensionnement. La faible complexité de ces unités ne supportant pas le surcoût lié à l'usage de matériel de test, on préfère utiliser qui est embarqué aux terminaisons des communications pour tester l'intégrité de ces dernières et donc les adaptateurs. La fonctionnalité de ces blocs étant statiquement définie, leur conception étant abordée au cas par cas, leur flexibilité est très restreinte.

2.3.5 Les ponts (*bridge*s)

Les ponts sont une application spécifique des adaptateurs à la connexion de plusieurs bus supportant chacun plusieurs maîtres. Ils doivent donc s'enquérir de :
– L'attribution du bus. En termes d'organisation, l'initiateur de la communication ou maître prend le contrôle de son bus et accède au pont se comportant alors comme un esclave. Le pont formule alors une requête pour obtenir l'accès au bus connecté à l'esclave et agit ainsi en tant que maître sur ce bus.
– La transformation de protocole dans le cas de bus hétérogènes ; Toutes données échangées par un maître sur son bus suivent un protocole d'échange qui peut ne pas être supporté par le bus connecté à l'esclave ciblé. Le pont doit alors adapter ces deux protocoles.

Ils sont généralement implémentés en logiques cablées. Leur utilisation est requise lorsque l'adaptation entre deux sous-réseaux de communication doit être opérée. Cet état de fait se présente lorsque le réseau est hiérarchisé afin de :
– offrir localement les performances requises ;
– accentuer le parallélisme en utilisant des ressources de transports concurrentes ;
– reporter les problèmes d'incompatibilité de protocoles des différents ensembles processeur/bus natif, sur un unique composant et permettre aussi loin que possible l'utilisation des protocoles natifs.

Pour découpler la synchronie des différents bus il est souvent fait appel à des mécanisme de mémoire tampon ou « buffer ». Les coûts et les performances relatifs à ces composants découlent donc des implémentations des machines

d'états exécutant les divers protocoles et gérant ces mémoires. Ces composants doivent supporter les bandes passantes propres à chacun des bus adapté afin d'optimiser l'usage de ces ressources partagées de communication. La monopolisation des deux ressources de calculs dépréciant fortement les performances globales, une attention particulière est portée sur les performances de ces goulots d'étranglement. La complexité de ces composants restant relativement faible, leur consommation énergétique est réduite. Cette simplicité fonctionnelle résulte aussi en une occupation surfacique héritée pour l'essentiel des mémoires tampons embarquées. Ces dernières étant dimensionnées pour ne contenir qu'un petit nombre de paquets sont très peu volumineuses. Comme pour les adaptateurs, lorsque les protocoles à adapter ne sont pas des plus coutumiers, il est nécessaire de développer des modèles spécifiques aux paires de bus considérées. Aussi, que l'on réutilise des composants préconçus ou que les conçoivent spécialement, il est nécessaire de dimensionner les mémoires tampons pour efficacement découpler les deux bus et ainsi diminuer les latences. La complexité de ces composants nécessite rarement l'usage de ressources de test. Ces composants répondent à une spécification bien précise : adapter deux ressources de communication en supportent une bande passante donnée. La seule flexibilité envisageable concerne la mise à l'échelle de la gestion des canaux concurrents connectés (ex : nombre de ligne d'un cross-bar).

2.4 Les architectures monoprocesseurs

Ces architectures que la littérature désigne par « Single Instruction stream, Single Data stream (SISD) » sont construites autour d'un unique processeur de logiciel agissant comme maître sur les composants (ou périphériques) qui lui sont associés.
Lorsque que ces composants apportent des accélérations locales de l'algorithme, ils sont appelés *coprocesseurs*. Il s'agit de petits processeurs matériels implémentant des fonctionnalités données qui nécessitent une exécution performante. Leur fonctionnement ainsi que les données à traiter sont dictés par le processeur de logiciel. Ce fonctionnement s'apparente à :
– lecture d'une configuration ou code d'opération à effectuer,
– réception des opérandes,
– calcul du résultat,
– assertion de la complétion au processeur,
– transmission du résultat au processeur.
Ces blocs esclaves fournissent une accélération certaine à l'exécution d'une infime mais très récurrente partie de l'application. Bien qu'étant des unités de traitement autonomes, les dépendances de données font qu'elles ne permettent pas une exécution concurrente de leur algorithme spécifique avec celle de l'algorithme applicatif exécuté sur le processeur de logiciel.
Ce schéma est utilisé pour les applications ayant un nombre limité de flux de données à traiter ainsi que contraintes temporelles faibles. L'usage d'un seul processeur permet de réduire le coût du système. De plus, il offre une utilisation améliorée des ressources de calcul. En effet, toutes les fonctionnalités étant concentrées en son sein, la gestion des ressources est aisée. Enfin, toutes les communications inter-fonctionnalités sont réalisées en interne et en logiciel affranchissant ainsi le système des coûts et latences des communications matérielles inter-processeurs. Cependant la disponibilité d'une unique ressource de contrôle interdit toute exécution concurrente de l'algorithme applicatif et les performances globales chutent drastiquement alors que la complexité du système s'accroît. La seule flexibilité de ce type d'organisation vient du processeur de logiciel dont le comportement peut être modifié à volonté. Il est malheureusement difficile d'étendre ses performances pour répondre à un accroissement de la complexité de l'algorithme ou du nombre de flux de données, sans dégradations conséquentes des performances générales. L'usage d'organisation comportant plusieurs unités de contrôle et/ou de calcul est alors requis pour le traitement d'un modèle parallélisé de l'application.

2.4.1 Interface Logicielle/Matérielle

Le terme *Interface Logicielle/Matérielle* désigne le couplage existant entre l'exécution du logiciel de contrôle des IO (les pilotes) par un processeur donné et le matériel de communication (c-à-d. coprocesseur et réseau de communication).
Le processeur se comporte comme maître vis-à-vis de ces périphériques (esclaves). En d'autres termes, pour tout

échange entre périphériques et processeur, c'est ce dernier qui en a l'initiative. Ceci implique qu'un processeur doit régulièrement consulter ses périphériques afin d'acquitter tout événement externe en des délais acceptables pour les contraintes temporelles de l'application.
Il peut pour cela procéder de deux façons différentes :

La scrutation ou attente active, consiste en lectures périodiques des registres d'états des périphériques afin de constater l'avènement d'une communication. Cette solution comporte un inconvénient de taille lorsqu'il s'agit d'interagir avec un grand nombre de canaux : les accès au matériel de communication requièrent l'intégralité des ressources du processeur, qui ne peut donc continuer à calculer l'algorithme applicatif. Bien entendu, il est possible de régler indépendamment pour chaque canal la période de scrutation, mais il faut pondérer cette solution par la latence de réponse du processeur à une communication, le nombre élevé de canaux et les contraintes «Temps-Réel» imposées à l'exécution des tâches.

Les interruptions matérielles sont des requêtes émises par les périphériques matériels afin de détourner le processeur de sont flot d'exécution courant. Ainsi lorsqu'un périphérique (par exemple un adaptateur de canal) veut avertir l'ensemble logiciel d'un changement de son état, il peut émettre une telle requête afin que l'unité matérielle, embarquée au cœur du processeur en charge de leur traitement puisse automatiquement lancer l'exécution d'un logiciel dédié capable de prendre en compte cet événement extérieur (lancer un pilote de communication). Ici la scrutation n'est plus opérée par le logiciel, mais par une fonctionnalité matérielle du processeur autorisant en parallèle l'exécution de l'algorithme applicatif. Cette solution présente deux surcoûts. Le premier, qui est relatif au coût matériel du mécanisme de scrutation, ne peut être tenu en compte car il est nécessaire à l'utilisation d'autres périphériques (temporisateurs, arbitre de bus, décodeur d'adresse). Le second est lié au délai de traitement des interruptions, c-à-d. le temps nécessaire pour détecter une requête puis interrompre le flot courant d'exécution puis exécuter la routine de traitement appropriée. Ce délai peut correspondre au temps d'exécution de plusieurs dizaines d'instructions (27 instructions pour le processeur ARM7TDMI d'après [2]).

Scrutation de registres

Dans la multitude de processeurs existants, on ne distingue que deux possibilités d'accès au matériel externe. Soit le processeur possède des unités de communication, embarquées dans son cœur, soit il doit accéder à de telles unités, externes cette fois-ci, de la même façon qu'il accède à sa mémoire locale. Les micro-contrôleurs illustrent la première catégorie. [21, 22] abordent cette approche. Mais cette méthode ne peut malheureusement pas répondre aux deux problèmes suivants :

1. La variété de protocoles utilisés pour les interfaces de composants matériels ne peut être couverte par l'utilisation de périphériques, monolithiquement associés au processeur aussi configurables soient-ils.
2. Le nombre de canaux parallèles de communication connectés à un même processeur étant en augmentation, les périphériques capables de les gérer pourraient rapidement se révéler être de disponibilité insuffisante.

La seconde catégorie regroupe tous les processeurs dépourvus d'unité matérielle de communication interne. Aussi doivent-ils procéder à des accès mémoires sur des adresses physiques fixées, leur permettant ainsi de communiquer avec des composants matériels attachés à leur bus mémoire. Mais d'un point de vue logiciel, deux variantes de ces accès existent (et peuvent même co-exister) :

1. Le jeu d'instruction du processeur offre l'usage d'instructions dédiées aux communications. Ainsi [43] dispose des instructions `in` et `out` permettant l'accès à des plages mémoires protégées et réservées pour les communications.
2. Les ressources de communication étant connectées au bus mémoire du processeur, ce dernier peut utiliser ses instructions de transferts entre registres et mémoires classiques, telles que les `mov` pour les processeurs SPARC[57] ou les `ld` et `st` pour les processeurs ARM[46].

Afin de pouvoir mettre en œuvre le plus grand nombre de canaux et protocoles de communication, l'utilisation de ressources de communication externes peut être adoptée. L'instruction exécutée par le processeur pour accéder à ces ressources est sans influence, seule l'activité des signaux du bus mémoire lors de ces cycles de lecture/écriture importe. Cependant une organisation en couches du logiciel telle que celle présentée dans [29] est un gage de fiabilité non-négligeable pour ces opérations d'entrées/sorties.

Mécanisme d'interruption

Lorsque le nombre de canaux de communication est important, leur gestion par scrutation cause un taux d'accès infructueux aux périphériques et des latences qui ne peuvent plus être tolérées. La maîtrise de la complexité et du volume de spécification d'un système électronique comportant un grand nombre de canaux étant un objectif de ces travaux, un schéma d'interaction à base d'interruptions matérielles est adopté.
La figure 2.5 illustre l'utilisation faite de ce mécanisme au sein des architectures ciblées. Ainsi l'interaction logiciel/matériel se décline en 6 étapes :

1. L'adaptateur de canal voulant attirer l'attention du processeur, émet un signal de requête d'interruption en direction de l'adaptateur de module ;
2. Le gestionnaire de requête embarqué dans l'adaptateur de module mémorise cette requête et émet une requête en direction du processeur.
3. Le processeur reconnaît cette requête puis, (au bout d'un temps rendu non déterministe par la gestion de priorités d'exécution assignées à chaque section de code logiciel ainsi qu'à chaque source d'interruption) en vient à la traiter ;
4. La routine de traitement alors lancée, opère un accès en lecture sur l'adaptateur de module, à une adresse spécifique ;
5. Ce dernier répond à cet accès par l'identifiant de l'adaptateur de canal le plus prioritaire, parmi tous ceux qui avaient émis une requête. Cet identifiant, un entier, est aussi appelé vecteur d'interruption car il permet de rechercher dans un tableau en mémoire programme, l'adresse de la routine de traitement spécifique à ce canal. L'utilisation des adresses de ces routines comme vecteur permet des performances plus élevées, mais implique que ces adresses soient connues ou figées ;
6. Cette routine pourra alors accéder à sa guise au canal par l'adresse physique qui lui a été assigné.

2.5 Les multi-processeurs

Ces architectures permettent l'exécution concurrente de plusieurs fonctionnalités. Mais il nous faut faire la différence entre l'exécution de plusieurs instances d'une même fonctionnalité sur des données différentes offerte par les SIMD, l'exécution de fonctionnalités différentes sur les mêmes données proposée par les MISD et les MIMD traitant des données différentes par des fonctionnalités diverses.
Une description de ces variantes est apportée par les trois paragraphes suivants.

2.5.1 Les SIMD

Les Single Instruction stream, Multiple Data streams (SIMD) sont des architectures multiprocesseurs ayant un flot d'instruction séquentiel et plusieurs flots de données parallèles. Leur usage est courant pour les applications de traitement du signal et de l'image. En effet elles permettent d'appliquer concurremment le même algorithme à plusieurs « fenêtres » de l'information (portions d'image, tranches temporelles d'un signal). Ces machines sont couramment appelées *processeurs vectoriels* et *processeurs matriciels* selon la nature du fenêtrage. Elles sont composées d'un nombre massif de processeurs identiques, appelés Process Unit (PU).
Cette organisation est souvent utilisée pour la conception d'un type de processeur de logiciel spécialisé dans le traitement du signal : le Digital Signal Processor (DSP)[108]. Elle est aussi utilisée, à une échelle plus importante, pour des applications de traitement de l'image [100]. Ces machines sont à même de fournir les meilleurs performances pour l'exécution d'un même traitement sur des vecteurs de données. En effet puisque chaque donnée du vecteur est manipulée par un même algorithme et puisqu'il ne dépend pas des valeurs de ces données alors il est possible de centraliser le contrôle des vecteurs d'opérateurs constituant la partie opérative. Les performances de ces machines sont en rapport direct avec le nombre de ressources de la partie opérative. Leur fonctionnalité est hautement spécifique à l'application.

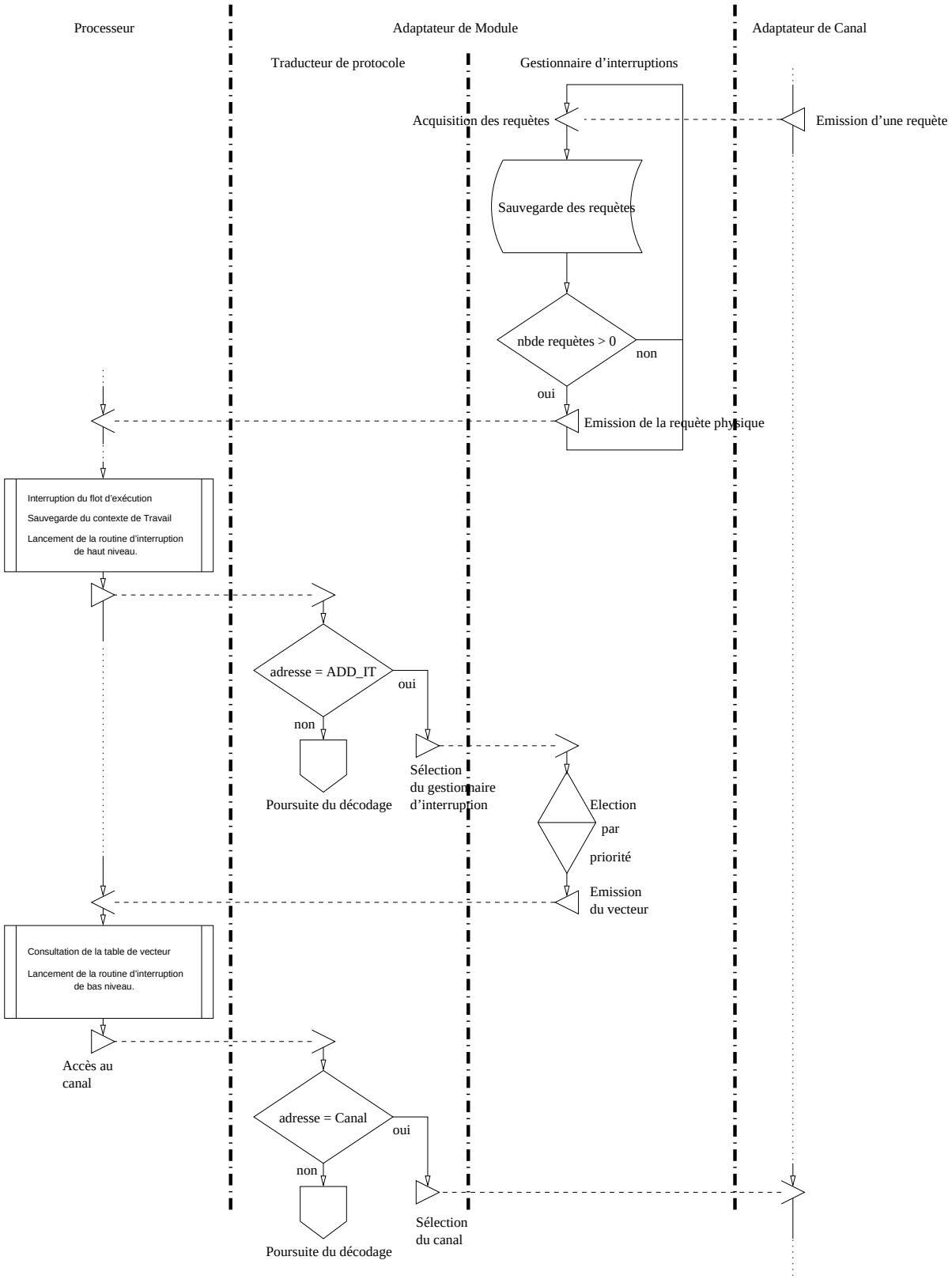


FIG. 2.5 – Communication HW/SW par interruptions.

2.5.2 Les MISD

Ce concept d'architecture met en œuvre l'exécution parallèle de plusieurs instructions sur un même flot de données. C'est pourquoi elles se nomment Multiple Instruction streams, Single Data streams (MISD). A ce jour, aucune implémentation de telles organisations n'a été recensée. Seul le mécanisme de *pipeline* utilisé dans les micro-processeurs (ou processeurs de logiciel) approche ce concept.

2.5.3 Les MIMD

Les Multiple Instruction streams, Multiple Data streams (MIMD) sont faîtes de multiples instances de processeur, ayant chacune son propre flot de données et son propre flot d'instruction. Chaque processeur a donc la possibilité de calculer un algorithme différent pouvant coopérer avec d'autres par envoi de messages et/ou partage de mémoire. Lorsque l'algorithme applicatif peut être parallèlisé, mais que cette décomposition n'est pas forcément régulière et l'exécution de ces branches n'est pas synchrone, alors l'utilisation de plusieurs unités de contrôle (et de leurs unités de calcul respectives) est requise. Enfin, elle peut résulter de l'implémentation d'un système par la composition des implémentations de ses sous-systèmes. Si les algorithmes exécutés sur chacun des processeur sont découplés et n'ont que peu de points de synchronisation en commun, alors l'utilisation de plusieurs unités de contrôle offre une accélération conséquente à l'exécution de l'algorithme globale. Lorsque le nombre de processeurs n'est pas limité par des contraintes d'implémentation (ex : un maximum de 16 processeurs autour d'un bus partagé AMBA AHB), l'organisation sera considérée « scalable ». Quant aux contraintes sur les types de processeurs supportés, elles sont discutées dans la section 2.7.

2.6 Organisation de la mémoire

Trois sous-catégories d'organisation sont distinguées en fonction de l'égalité des processeurs dans les accès aux mémoires. Il s'agit 1º des accès uniformes (UMA), 2º des accès cachés (COMA) et 3º des accès non-uniformes (NUMA).

2.6.1 Les UMA

Les Uniform Memory Access (UMA) offrent un accès équitable aux mémoires à tous les processeurs. Leur structure est un assemblage de plusieurs instances de processeurs et d'un bloc de mémoire partagé et centralisé, autour d'une ressource de communication. Celle-ci peut être un bus partagé (ex : AMBA[1]) ou un réseau commuté (ex : SPIN32[33]).
Un plan mémoire principal et global est associé aux unités de mémorisation et de communication. Il est découpé en tranches régulières chacune associée à un processeur. Un mécanisme matériel et transparent de translation des adresses peut être utilisé pour abstraire ce découpage auprès de chaque processeur. Ces modèles sont utilisés pour les calculs parallèles sur des données volumineuses qui peuvent alors être échangées par mémoire partagée. Aussi les retrouve-t'on dans les applications de traitement d'image. Le coût de cette solution est une affine du nombre de processeurs.
Pour les performances, la règle n'est pas aussi linéaire. En effet, les performances des communications peuvent ne pas être étendues, plus particulièrement dans les cas où un bus partagé est utilisé. La souplesse de ce modèle provient de la facilité à reconfigurer le comportement des processeurs par une programmation de la mémoire programme partagée. La mise à l'échelle des performances est malheureusement limitée par les performances des ressources de communication.

2.6.2 Les NUMA

Les Non Uniform Memory Access (NUMA) doivent leur existence au point suivant : puisqu'au sein d'une UMA, le réseau de communication est un goulot d'étranglement hiérarchisons les UMA pour localiser les utilisations des ressources de communication. Ainsi une NUMA est un assemblage de grappes de processeur autour d'un réseau de communication. Chaque grappe peut être vue comme une architecture SISD (ou MIMD en fonction du nombre de processeur par grappe) avec une organisation locale UMA. La mémoire qui est distribuée dans chacune de ces grappes est partagée par tous les processeurs de la grappe, mais aussi par les autres grappes.

Il en résulte une hiérarchisation des ressources de communication en un bus système connectant les grappes qui contiennent chacune un bus local. De ce fait, les accès mémoire sont hétérogènes puisqu'un processeur peut facilement utiliser la mémoire connectée sur le même bus local et bien plus péniblement celle appartenant à une autre grappe.
Le principal défaut des NUMA étant la variation des délais d'accès à la mémoire suivant qu'elle soit locale ou éloignée (dans une autre grappe), les Coherent Cache Non Uniform Memory Access (CC-NUMA) cherchent à limiter cet effet en utilisant des caches cohérents qui assurent la recopie automatique des données partagées par plusieurs grappes. Le projet Directory Architecture for SHared memory (DASH)[52] (Fig. 2.6) de Stanford et [58] mettent en œuvre cette organisation.

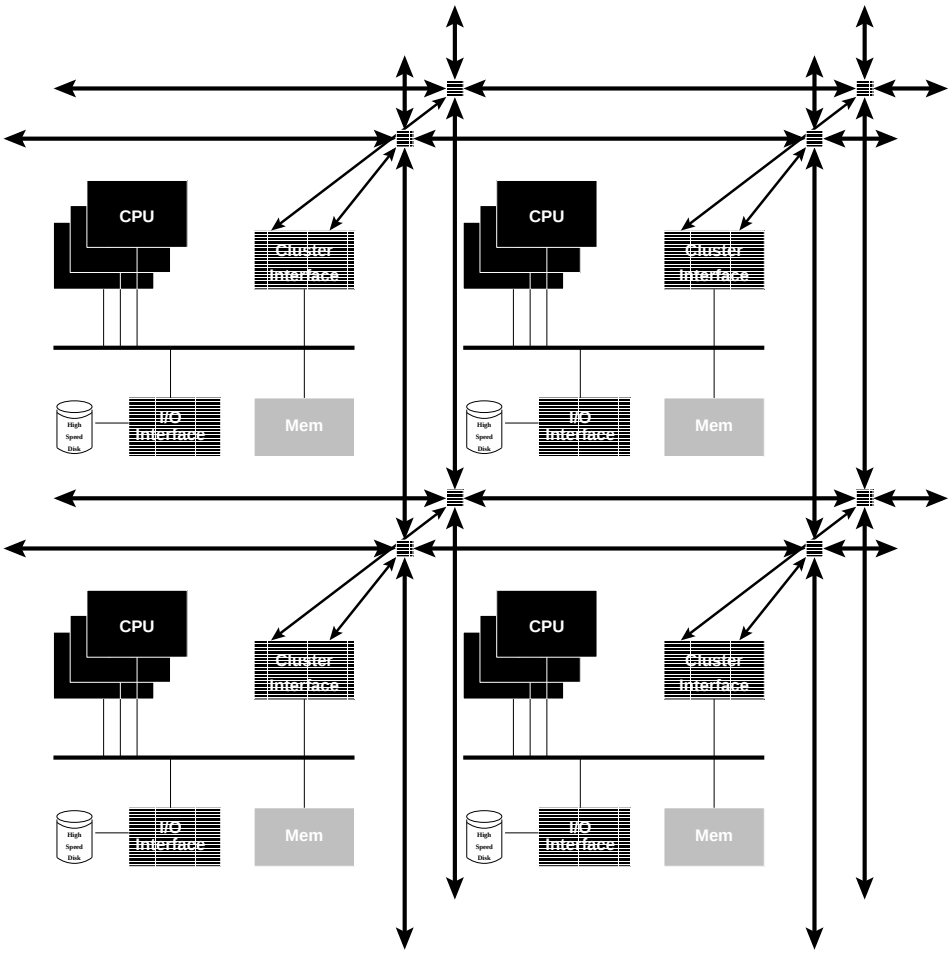


FIG. 2.6 – Architecture multi-ordinateurs DASH.

Dans cette solution, un répertoire est utilisé dans chaque nœud de calcul pour colorer les copies locales de données modifiées afin que les accès futurs à des copies distantes de ces mêmes données déclenchent une mise à jour. Le matériel nécessaire à l'implémentation des répertoires et des bus de cohérence est très coûteux pour des systèmes sur puces. Le coût de ces architectures étant fortement corrélé au nombre de grappes, il contraint très fortement la mise à l'échelle de ces architectures.
Les No Cache Non Uniform Memory Access (NC-NUMA) sont des implémentations des NUMA sans accélération des accès aux mémoires distantes par l'utilisation de caches. Ici ce sont des mémoires dont le contenu est mis à jour sur demandes explicites des processeurs qui jouent partiellement le rôle de cache. Ces mémoires sont communément appelée « scratch-pad ». L'architecture FLASH[47] et les CM* de Carnegie-Mellon sont des exemples de cette organisation.
Leur coût est très diminué par l'absence des mémoires caches à gestion globale, mais surtout par l'usage d'un réseau de communication beaucoup plus léger que les bus de cohérence. Flash utilise des connexions point-à-point et une topologie matricielle. Le bloc `MAGIC` agît comme un serveur de données, ses clients pouvant être le processeur local où tout autre bloc `MAGIC` distant.

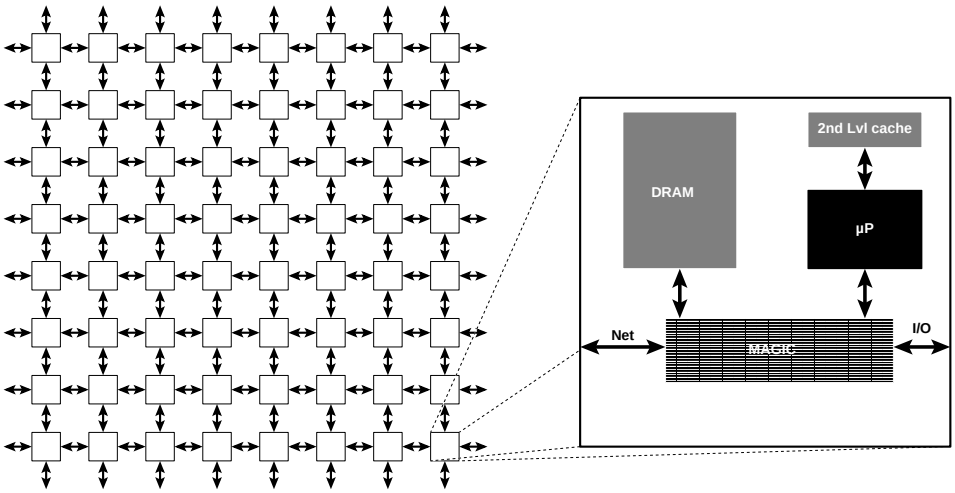


FIG. 2.7 – Architecture multiprocesseur FLASH.

2.6.3 Les COMA

Les Cache Only Memory Access (COMA) ont la particularité de distribuer l'implémentation de la mémoire partagée dans les caches de chaque processeur. Ainsi les informations sont échangées par recopie de lignes d'un cache à un autre. Les caches sont très chers, d'intégration très faible donc limités à de faible capacité. De plus le bus de cohérence réclame à lui seul une implémentation hors de prix (Cross-bar). Mais ce sont les MIMD les plus performantes.

2.7 Régularité des architectures

La régularité d'une architecture est l'identification dans sa structure de la reproduction d'un schémas ou motif à base de composants de calcul. Nous ne distinguerons que trois niveaux de régularité : 1^o les architectures symétriques, 2^o les architectures régulières et 3^o les architectures hétérogènes.

2.7.1 Les SMP

Il s'agit de cas particuliers de l'organisation MIMD, car les Symetric Multi-Processor (SMP) utilisent des instances d'un même processeur interconnectées selon un schéma régulièrement répété. Ces architectures sont utilisées comme plateformes génériques pour l'implémentation d'applications diverses. Elles sont utilisées pour des systèmes aux contraintes faiblement stressantes et pour une production de faible volume. En effet l'utilisation d'une mémoire partagée globale (UMA) permet de découpler la localisation des données de la localisation des unités de calcul (processeur de logiciel assigné). Cette possibilité permet d'équilibrer la charge ou le taux d'utilisation des processeurs et donc d'obtenir de meilleures performances. La méconnaissance des applications ciblées pousse à répondre aux besoins en performances par la mise à disposition d'un nombre élevé de ressources de calcul. Ne bénéficiant d'aucune métrique, les concepteurs de ces architectures ne peuvent que choisir et assembler symétriquement ces ressources. Mais cette généricité les rendent inadaptées aux applications monopuces. En effet, n'offrant pas des média localement spécialisés à la réalisation de communications, leur réseau d'interconnexion est un goulot d'étranglement et ces architectures doivent être surdimensionnées afin de supporter les bandes passantes élevées localement requises par l'application. Il en va de même pour les ressources et performances de calcul.
La symétrie se révèle avantageuse pour la mise à l'échelle d'une architecture, puisque la régularité guide l'ajout de nouvelles composantes.
Quant à la flexibilité comportementale, celle-ci se limite à la fonctionnalité du motif régulièrement instancié. Si ce motif est à base de processeurs de logiciel, alors la flexibilité de ceux-ci garantit la flexibilité de l'architecture.
Dans de telles architectures, le nombre réduit de types de composants utilisés encourage le développement spécifique de ressources de communication compatibles avec chacun des composants connectés. Le besoin d'adaptateurs est alors moins ressenti. Cependant si l'assignation des processeurs aux tâches est dynamique (ex : l'équilibrage de charge), il est nécessaire de pouvoir translater au sein de la mémoire partagée, de manière transparente les plans mémoires locaux

à chaque processeur. On insère alors entre les processeurs et les ressources de communication, des *MMU*, pouvant être assimilés à des interfaces.

2.7.2 Les architectures régulières

Ces architectures ont une topologie régulière, c'est-à-dire un assemblage systématique des composants autour du réseau de communication. Cependant, chacun de ces composants est de nature et de constitution libres.
Un exemple d'une telle architecture est présentée en sous-section 2.8.2.
Ces architectures sont issues de la spécialisation des nœuds de calculs aux fonctionnalité qui leur incombent, tout en utilisant des ressources de communication génériques. Cette solution basée sur une première approche de spécialisation offre une meilleure adéquation des coûts de ressources utilisées aux exigences en performances de l'application. De plus, elle permet une systématisation ou automatisation de l'assemblage des composants. Malheureusement, ces réseaux de communication génériques deviennent les nouveaux goulots d'étranglement face à l'accroissement des exigences en performances globales des applications. Des solutions à base de topologie matricielle sont alors envisagées comme l'illustre la puce *PicoArray* de *PicoChip*[7] avec ces 430 processeurs RISC 16 bits faiblement hétérogènes organisés en matrices de grappes de 4 processeurs chacune.

2.7.3 Les architectures hétérogènes

Il s'agit de l'anti-thèse des SMP : Il n'existe pas de motif unique reproduit pour constituer l'architecture. Des ressources spécifiques sont mises en œuvre pour localement améliorer l'adéquation entre l'implémentation et les fonctionnalités exécutées. Partout où une bonne adéquation entre l'implémentation et la fonctionnalité est nécessaire dans les applications ayant un marché à fort volume, l'utilisation d'architecture hétérogène est préconisée.
L'hétérogénéité permettant de spécialiser l'implémentation, il est plus aisé d'équilibrer les coûts de fabrication et consommation avec le niveau de performance. De telles architectures seront par conséquent plébiscitées pour les applications monopuces. Cependant la diversité des composants nécessite une étendue de compétences et multiplie les efforts de conception. Si l'irrégularité gène l'ajout de nouveaux composants, la possibilité de spécialiser ceux-ci autorise des gains substentiels en performance.

2.8 Quelques exemples d'architectures multiprocesseurs monopuces

Quatre architectures de systèmes monopuces issues de l'industrie sont d'abord étudiées dans les premières sous-sections. Puis leurs caractéristiques seront synthétisées afin de préparer la mise en place d'un modèle générique dans la sous-section 2.8.5.
Il existe autant d'architectures que de systèmes sur puces, auxquelles viennent s'ajouter les architectures « classiques » issues de la littérature [28, 25, 36, 40, 49, 96]. Cette section ne prétend pas en adresser l'ensemble. Son but est uniquement d'analyser des exemples d'architectures de systèmes sur puces qui semblent avoir été réalisés par assemblage de composants et dont les processus de conception sont plus ou moins enclins à leur automatisation.

2.8.1 L'architecture « Prophid » de Philips

L'architecture « Prophid »[50] est issue de la recherche de Philips sur les architectures pour des applications multimédia. La figure 2.8 présente la structure de cette architecture. Les architectures « Prophid » sont composées d'un cœur de processeur logiciel RISC (*CPU*) et d'accélérateurs matériels pour le traitement du signal (*ADS*). Le nombre de ces derniers est évolutif.
Les accélérateurs communiquent entre eux au moyen d'un bus « *knockout* » et d'une mémoire partagée. Le bus « *knockout* » est issu de l'association d'un bus cross-bar, de multiplexeurs, de décaleurs et de FIFO. Il est schématiquement représenté par la figure 2.9.
Les blocs mémoires se réduisent à une mémoire DRAM partagée et deux petits blocs SDRAM servant de caches pour le processeur logiciel.
L'organisation de cette architecture est de type maître-esclave. Les blocs *ADS* ne sont que des esclaves du processeur logiciel mais opèrent des fonctionnalités qui sont propres à chacun d'entre-eux. Cette architecture appartient à la famille des MIMD. L'organisation mémoire consiste en une mémoire unique partagée et centralisée à accès non-uniforme

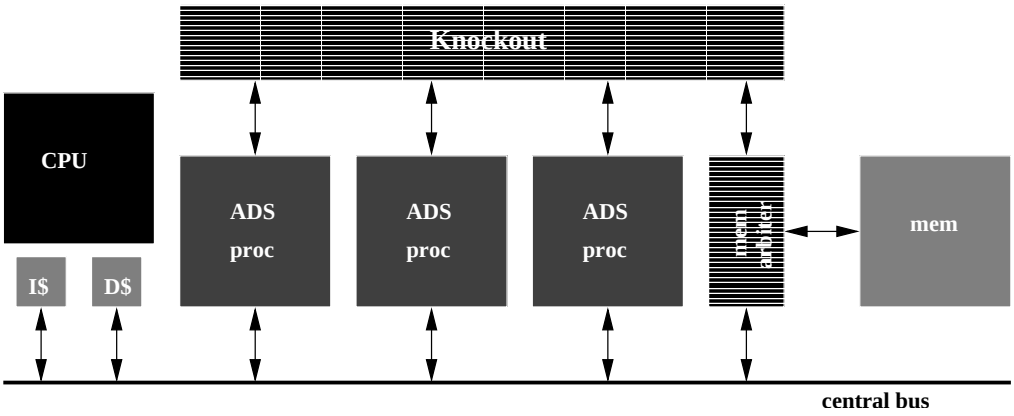


FIG. 2.8 – Architecture Prophid de Philips.

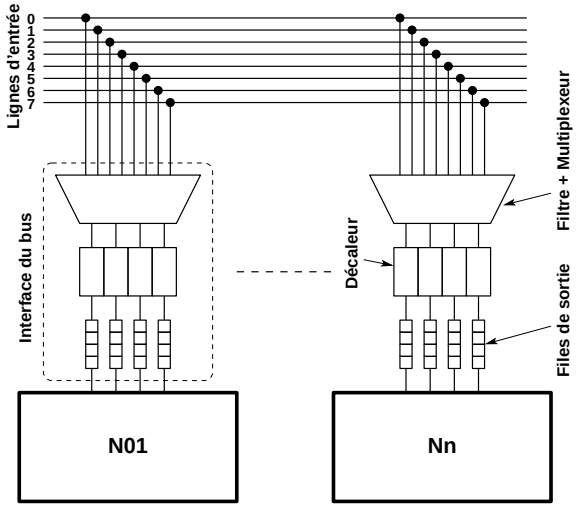


FIG. 2.9 – Bus « *knockout* ».

(NUMA) car « *CPU* » ne peut y accéder par le bus « knockout » qui offre un débit plus élevé et des latences plus faibles que le bus partagé. Bien que les blocs « *ADS* » puissent être réalisés par des composants matériels ou des coprocesseurs programmables pouvant exécuter du logiciel, les « *ADS* » présentent tous la même interface. Ainsi cette architecture est hétérogène et fortement régulière. La symétrie des accélérateurs, l'usage d'un bus « Crossbar » permet de présager des extensions aisées de cette architecture par l'ajout d'accélérateurs *ADS* et de lignes au « Crossbar ».

2.8.2 La Set-Top Box « Viper pnx8500 » de Philips

Le *Viper pnx8500* ou *Nexperia* est un système monopuce développé par Philips afin de répondre au marché des télévisions numériques (« *Set-Top Boxes* »). Cette architecture monopuce est décrite dans [74, 26]. Il s'agit d'une architecture permettant la gestion de flux multimédia pour la télévision numérique. Ses fonctionnalités permettent la décompression audio-vidéo, la surimpression d'images et l'interaction utilisateur-opérateur télévisuel. La structure du *pnx8500* est illustrée par la figure 2.10.

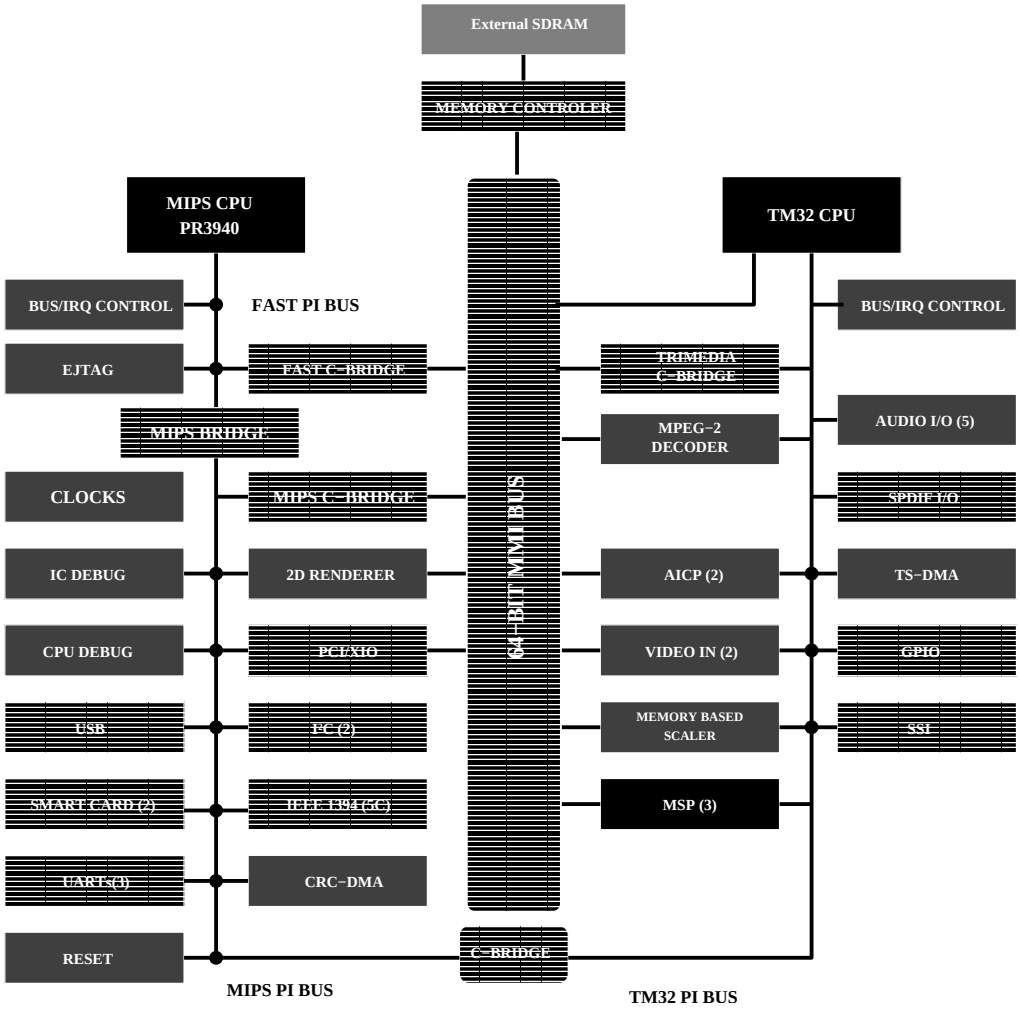


FIG. 2.10 – Structure du *Viper* de *Philips*.

Ses caractéristiques principales sont directement héritées des architectures « Prophid » présentées précédemment. En terme de calcul, cette architecture se compose de cinq processeurs logiciels et une demi-douzaine d'accélérateurs matériels sont utilisés. On retrouve pour les processeurs logiciels un cœur RISC *MIPS* de 32 bits, un cœur VLIW *Trimedia* de 32 bits et un cœur RISC 16 bits par unité *MSP* (2 ou 3 instances).
Parmi les processeurs matériels, on remarque des accélérateurs de traitement d'images (*2D GRAPHICS*, *AICP*et *MEMORY BASED SCALER*), un décodeur de flux vidéo (*MPEG-2 DECODER*), des unités dédiées au test (*EJTAG*, *IC DEBUG* et *CPU DEBUG*) et des composants de manipulation de blocs mémoires (*CRC-DMA* et *TS-DMA*).
La structure des communications est une topologie complexe. Trois bus partagés de type PIbus[69] sont utilisés pour

les communications processeurs↔périphériques essentiellement utilisées pour le contrôle. Pour les accès à la mémoire, un réseau « knockout » (*MMI BUS*) est utilisées afin d'en réduire l'étranglement. Cinq ponts sont utilisés afin de prolonger les accès d'un PIbus vers un autre ou vers les connexions point-à-points. Des interfaces analogiques (*VIDEO IN, AUDIO I/O* ...) et numériques (*USB, I^2C* ...) avec l'environnement du système sont discernables. Des interfaces internes (non représentées) sont utilisées pour adapter les instances de cœurs préalablement conçus aux différents bus. Trois types de mémoires sont utilisées : 1^o un ensemble de blocs de mémoire SDRAM partagés et externes (mémoire « off-chip »), pouvant être accédés concurremment, 2^o des mémoires SRAM pour les mémoires locales ou caches des processeurs logiciels (non représentées) et 3^o de la mémoire ROM (non représentée) embarquée dans *RESET* pour l'initialisation du système.
L'organisation de cette architecture est une hiérarchie de deux grappes, chacune constituée d'une organisation maître-esclave autour d'un processeur logiciel. La couche logicielle de bas niveau (*Firmware*) enrichi cette organisation par une relation de dominance de la grappe du processeur *MIPS* sur celle du *TriMedia*. En effet, le processeur *MIPS* est chargé, en plus du contrôle de sa propre grappe, du contrôle de l'ensemble du système et peut réclamer des services de traitement du signal au *TriMedia*. Ces services sont exécutés à la demande du *MIPS*, et peuvent nécessiter l'utilisation des accélérateurs placés directement sous le contrôle du *MIPS*. Cette architecture permet l'exécution concurrente de plusieurs algorithmes constituant plusieurs flux de données (Interaction Homme-Machine, Audio, Vidéo). Cette architecture peut donc être classée parmi les MIMD. Elle utilise une mémoire partagée, centralisée comme unique unité de stockage des données partagées, l'accès à cette dernière est uniquement réalisable au travers du réseau « knockout ».
Le domaine d'application cible de cette architecture est le multimédia. Cette architecture est hétérogène à plusieurs niveaux : les processeurs logiciels sont hétérogènes, les accélérateurs sont hétérogènes et la communication est elle-aussi, hétérogène. Malgré toutes ces hétérogénéités, cette architecture est régulière. Les accélérateurs sont connectés selon un même schéma à une grappe locale axée sur un *PI-Bus*. Les grappes sont symétriquement regroupées de parts et d'autres du réseau « knockout » (*MMI*). Les performances requises ont poussé à une haute adéquation de l'architecture matérielle avec le découpage des fonctionnalités. L'utilisation de composants spécifiques pour implémenter ces fonctionnalités suffit à caractériser cette architecture d'hétérogène.
Cette architecture illustre parfaitement l'approche de conception par la composition de sous-systèmes. On distingue nettement deux sous architectures, l'une centrée sur le *MIPS* et l'autre sur le *TriMedia*. Certains blocs d'accélération tels que les *MSP* ont eux-mêmes une architecture évoluée. La quasi-totalité de ces composants a été réutilisée (non conçue spécifiquement pour cette application) et des adaptations de leurs interfaces matérielles et parfois aussi logicielles (pour les processeurs) ont dû être spécifiquement conçues afin de les connecter aux ressources de communication.

2.8.3 Le processeur « MAJC » de Sun

Le processeur logiciel « MAJC » de Sun est un Very Long Instruction Word processor (VLIW). Ce processeur a été spécialement conçu pour l'exécution de multiples tâches JAVA implémentant des fonctionnalités génériques ([98]). Sa structure est représentée sur la figure 2.11.

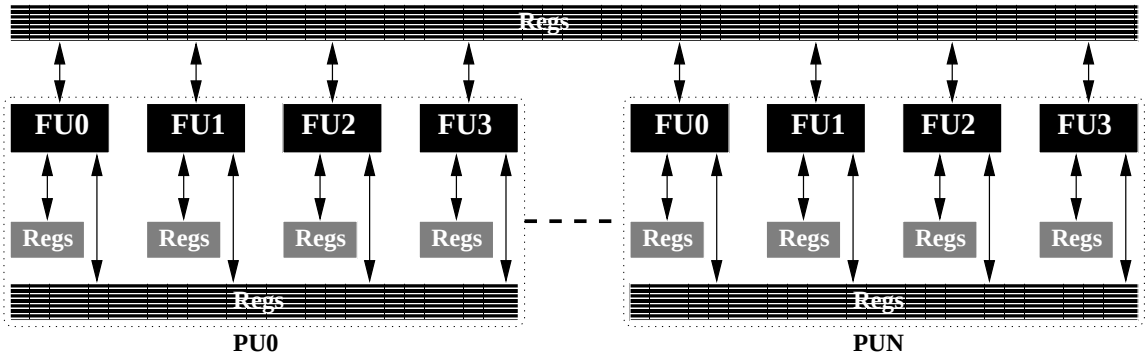


FIG. 2.11 – Architecture des processeurs MAJC

Cette architecture détaillée dans [87], peut contenir jusqu'à quatre processeurs logiciels ou Process Unit (PU) contenant chacun quatre unités fonctionnelles ou Functional Unit (FU). Chaque PU est capable de décoder et de traiter le sous

groupe d'instructions qui lui est destiné. Les seuls composants de communication présents sont des registres de SRAM hiérarchiquement partagés. Cette structure permet les communications entre les FU d'un même PU et inter-PU. Les registres précédemment cités font bien entendu partie des composants de communication, mais il faut leur associer des registres locaux à chaque FU.
Chaque FU peut traiter de façon autonome le décodage et l'exécution d'instructions. Cette architecture est de type MIMD car capable de traiter concurremment plusieurs algorithmes sur des flots de données distincts. Les accès mémoires ne sont pas homogènes car l'utilisation des registres partagés est restreinte à un groupe de FU. On se trouve donc en présence d'une organisation NUMA. La régularité et l'homogénéité de cette architecture sont extrêmes puisque seules des instances d'une même unité fonctionnelle sont symétriquement utilisées et la hiérarchie des registres partagés forme un arbre équilibré. Il s'agit manifestement d'une SMP.
Sun propose (figure 2.12) l'association de deux ou quatre « MAJC » ([99, 87]) organisés en CC-NUMA pour accroître le parallélisme.

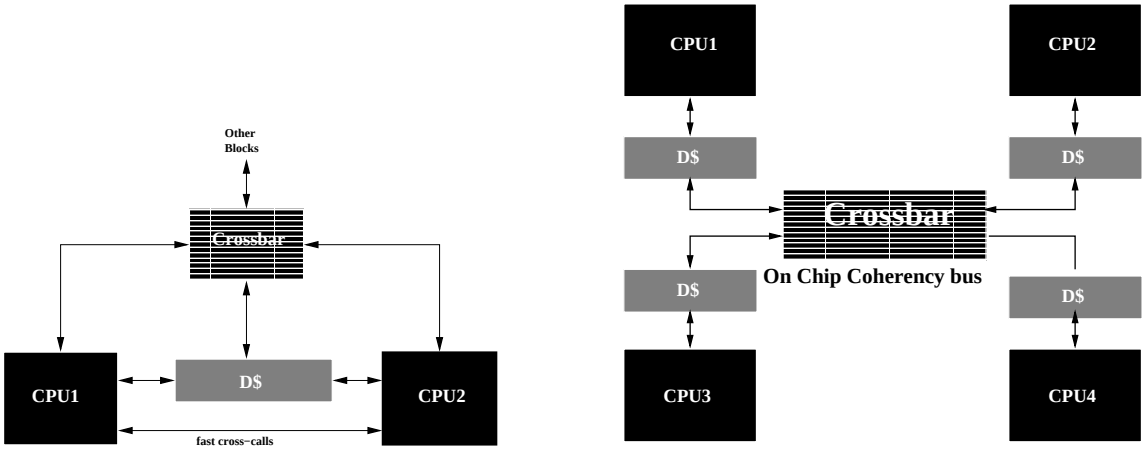


FIG. 2.12 – Architectures multiprocesseurs MAJC

2.8.4 Le processeur de jeux « Emotion Engine » de Sony

Le « Emotion Engine »[93] est le processeur de jeu utilisé dans la console de jeux « Playstation 2 » de SONY.
La figure 2.13 illustre la structure de cette puce.

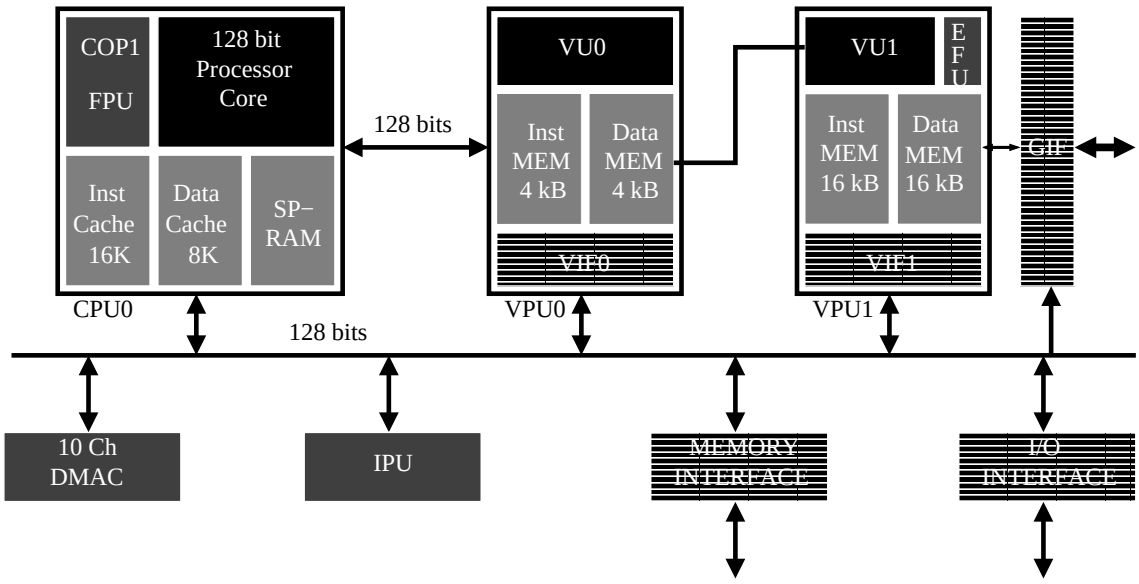


FIG. 2.13 – SONY Emotion Engine

Trois cœurs de processeurs logiciels sont discernables : 1^o un processeur MIPS III (`Processor Core`), de type RISC 128 bits ayant deux UAL entières de 64 bits associées à un coprocesseur pour l'arithmétique des nombres flottants ; 2^o un super-scalaire (`VU0`) intégrant quatre Floating Multiplier-Accumulator (FMAC), un Floating Divider (FDIV), une unité de décalage et une UAL ; 3^o un autre super-scalaire (`VU1`) très semblable à `VU0` (`VU1` est pourvu d'une Elementary Functional Unit (EFU) qui grâce à la combinaison d'un FMAC et d'un FDIV accélère sensiblement le calcul géométrique utilisé pour la synthèse graphique et les services apportés par `VIF1` sont plus évoluéés que ceux proposés par `VIF0`.).
Les processeurs matériels sont quant à eux réduits à un contrôleur de DMA (`DMAC`) pouvant gérer 10 canaux et un accélérateur de contrôle du protocole IP (`IPU`).
L'ensemble des composants est connecté à un bus partagé. On dénote aussi la présence de deux connexions point-à-points reliant deux à deux les processeurs logiciels. On compte aussi des blocs d'interfaçage tels que :

I/O INTERFACE : une unité centralisant toutes les entrées/sorties avec l'environnement sauf la sortie vidéo. Elle permet la connexion à une autre puce chargée du contrôle des entrées/sorties ;

VIF0 : un adaptateur permettant la connexion du processeur logiciel `VU0` au bus système, mais aussi le découpage/composition des mots de 128 bits du bus en flottants double précision (64 bits) directement utilisables par les FMAC de `VU0` ;

VIF1 : comme `VIF0` cette interface adapte son processeur (`VU1`) au bus système, mais est aussi capable de reconnaître et analyser des structures de données contenant à la fois des instructions et des données pour le traitement d'images en 3D. `VIF1` doit donc extraire les instructions de cette structure pour les recopier en mémoire cache programme et réciproquement pour les données en mémoire cache donnée.

GIF : le flux vidéo nécessitant une bande passante élevée et des fonctionnalités plus complexes, l'interface `GIF` lui est dédiée.

Sept blocs de mémoire SRAM se distinguent, trois sont associés au MIPS, les quatre autres sont associés par paires aux deux super-scalaires.
Le processeur logiciel MIPS III règne en maître sur cette architecture. Il peut déclencher des services sur `VU0`, `VU1` et même nourrir (indirectement en utilisant `DMAC`) ce dernier en instructions et en données. Ainsi le MIPS contrôle et synchronise l'exécution du jeu, il peut aussi se charger du traitement du flux audio. `VU0` est utilisé pour le calcul d'images 2D telles que l'arrière plan de scènes. Le calcul des objets 3D du premier plan est réalisé par `VU1`. Ils peuvent efficacement être superposés sur l'arrière plan grâce à la connexion point-à-point entre `VU0` et `VU1`, cascadant ainsi le traitement du flux vidéo. Nous nous trouvons donc en présence d'une architecture MIMD hautement spécialisée pour son domaine d'application.
L'organisation de cette architecture est de type NUMA. La présence de mémoires locales partagées (« publiques »), l'utilisation d'un réseau de connexions point-à-points non-complet (le MIPS n'est pas relié à `VU1`) suffisent à justifier cette classification. L'hétérogénéité de cette architecture n'est pas à démontrer, la variété des processeurs utilisés y suffit. Quant à sa régularité, aucune reproduction de motif ne peut être reconnue dans la topologie des interconnexions.
Ce processeur de jeu atteint 6,2GFLOPS et calcule de 16 à 66 Millions de polygones par secondes (en fonction des effets graphiques traités : rayons de lumières, brouillard et courbes de Bezier)malgré un cadencement « limité » à 300Mhz. Ces performances sont dues au cascadage du traitement vidéo, mais aussi à l'intégration de services dans les interfaces (`VIF0`, `VIF1` et `GIF`).

2.8.5 Analyse et architectures ciblées

Ces architectures (*MAJC* mise à part) de systèmes sur puces sont composées d'un assemblage de sous-ensembles spécialisés et ré-utilisés. Il résulte de cette spécialisation de l'architecture une hétérogénéité des composants (de calcul, de mémorisation, mais aussi de communication). Cette hétérogénéité peut localement créer une inadéquation entre un composant de calcul et les composants de communication auxquels il est raccordé. L'utilisation d'adaptateur peut être requise pour connecter les composants de calculs aux composants de communication. Enfin, toutes les architectures vues en exemples ainsi que les modèles classiques, toutes ont une structure qui peut être exprimée en trois couches. La première de ces couches comprend les composants de calcul et les composants de mémorisation. La seconde regroupe des interfaces qui permettent la connexion des composants de cette première couche aux composants de communication qui constituent la troisième et dernière couche.

Les architectures ciblées par cette thèse doivent couvrir l'ensemble des caractéristiques illustrées par les exemples précédents. Ces architectures sont généralement constituées d'éléments spécifiques pour efficacement répondre à l'implémentation d'une ou d'un ensemble d'applications. De plus, certaines applications demandent tellement de performances en calcul et en communication, que plusieurs instances de ces composantes peuvent être employées. Pour cela la combinaison de plusieurs instances de différents processeurs logiciels, de différents accélérateurs matériels, mais aussi de différentes ressources de communication doit être adressée. Enfin cette combinaison ne doit pas être restreinte par un schéma d'interconnexion ou d'organisation. Ceci se résume en le support d'**architectures multi-processeurs MIMD NUMA hétérogènes en composants de calcul, en composants de mémorisation et en ressources de communication**.

2.9 Modèle générique pour la représentation des architectures hétérogènes

Cette section présente un modèle couvrant la représentation des architectures précédemment étudiées. Les caractéristiques de ces architectures seront dans un premier temps rappelées, puis notre modèle générique d'architectures sera détaillé.

2.9.1 Architectures modélisées

L'hétérogénéité des architectures ciblées présentent quelques particularités.
Chaque instance de composant de calcul ou de mémorisation peut communiquer avec son environnement en utilisant une interface et des protocoles qui lui sont propres.
Le réseau de communication est lui-même constitué de ressources hétérogènes. Chacune de ces ressources (bus, réseau commuté, etc.) peuvent se distinguer par le support d'un protocole particulier régissant l'accès aux média, ainsi que le transport des données sur ces derniers.
De plus, le nombre de combinaisons possibles parmi cet ensemble d'instances croît exponentiellement avec leur nombre. En effet un même composant de calcul peut être connecté à plusieurs réseau de communication (tandis que la réciproque est triviale).
Afin de supporter ces caractéristiques, un modèle générique devra offrir **modularité**, **flexibilité** et **mise à l'échelle**. La **modularité** permet, par le cantonnement (circonscription) des spécificités d'une instance, de découpler un sous-ensemble de fonctionnalités du reste du système. La **flexibilité**, par le support de composants de domaines différents tels que numérique, analogique, Radio-Fréquence (RF)et Micro Electrical and Mechanical System (MEMS), autorise la représentation de systèmes complets à composantes hétérogènes.La **mise à l'échelle**, en ne contraignant pas le nombre de ressources composant le modèle, en garantie la pérennité face à l'accroissement du nombre de composantes dans le but de répondre aux besoins en performances.

2.9.2 Modèle générique d'architectures

Un modèle générique d'architecture supportant les caractéristiques précédemment citées est présenté dans [6] et est représenté par la figure 2.14. Sur cette figure, trois types de nœuds sont utilisés. Ce sont 1^o des processeurs de logiciel (`CPU`), 2^o des processeurs matériels (`IP`) et 3^o des mémoires (`MEM`). De même, les interconnexions sont composées d'un nombre non-contraint de média supportant des protocoles divers.
La **modularité** est obtenue par le découplage des unités de calcul/mémorisation des ressources de communications, grâce à l'adaptateur interfaçant chaque nœud de calcul/mémorisation au réseau de communication et par l'utilisation de ces derniers comme ressources exclusives de communications ; La **flexibilité** par la modélisation de toute nature de composant pourvu que les adaptateurs autorisent leurs connexions ; La **mise à l'échelle** repose sur la capacité de chaque adaptateur à supporter un nombre quelconque de connexions et sur la possibilité de composer régulièrement ces architectures en masquant les spécificités des composantes de calcul/mémorisation/communication par l'usage de ces adaptateurs.

L'Architecture Locale

La figure 2.15 illustre l'implémentation d'un nœud de calcul et sa connexion au réseau de communication.
L'implémentation est réalisée au travers d'une instance de processeur (`CPU core`), connectée au bus, aux instances de

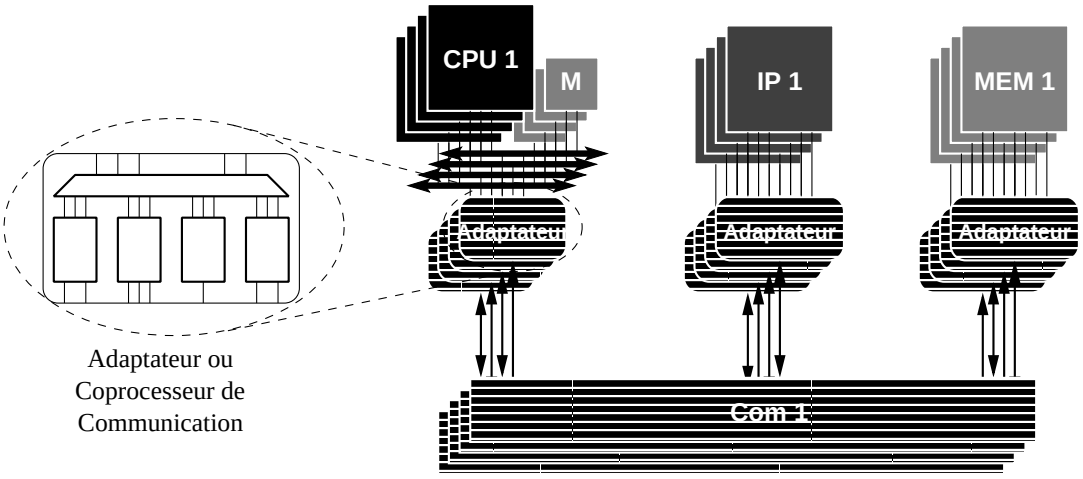


FIG. 2.14 – Modèle générique d'architecture.

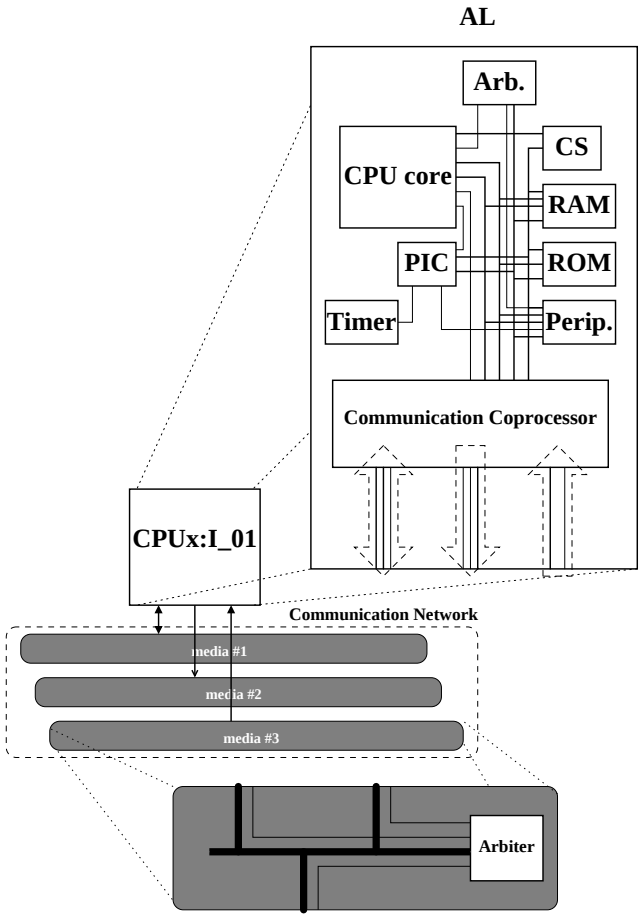


FIG. 2.15 – Architecture locale

mémoires (`RAM`, `ROM`) et autres composants nécessaires à son fonctionnement (gestionnaire d'interruption `PIC`, décodeur d'adresse `CS` et arbitre de bus `Arb.`).
Cette portion de l'architecture centrée sur un processeur est dénommée *Architecture Locale (AL)*. La connexion de cette AL aux média du réseau de communication est réalisée par un « **coprocesseur de communication** ». Les fonctions et composantes de cette charnière architecturale sont décrites par la sous-section suivante.

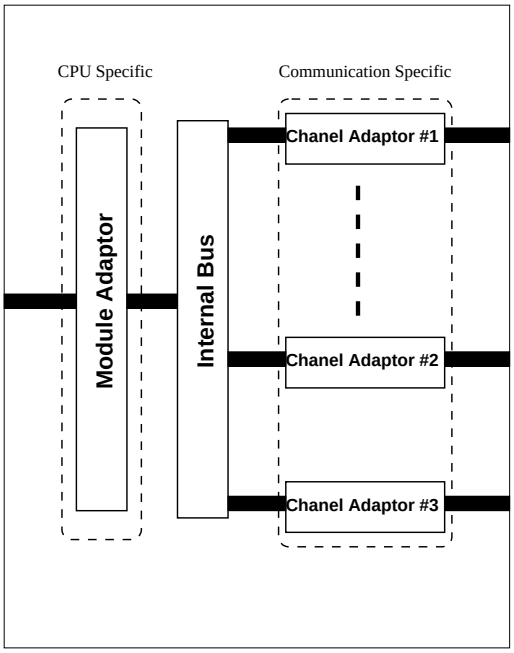


FIG. 2.16 – Coprocesseur de Communication (CC).

Le Coprocesseur de Communication En général les processeurs classiques ne connaissent qu'un accès unique vers le reste de l'architecture. Il s'agit du bus mémoire qui leur est propre et leur permet d'accéder aux mémoires qui leur sont associées. Chaque processeur doit donc traduire ses communications en autant d'accès mémoires que nécessaires au contrôle (protocole) de la communication et aux transferts des données. Le *Coprocesseur de Communication (CC)* est un concept général permettant de découpler le calcul et la communication. Sa structure est générique et peut être déclinée sous différentes solutions (cf. [111, 30]). Le CC peut prendre en charge 1^o la traduction du protocole d'accès mémoire du processeur en un autre plus apte à des communications inter-AL, 2^o la gestion du contrôle de ce protocole. Un exemple d'implémentation d'un coprocesseur, mettant en œuvre plusieurs canaux de communication est proposé en figure 2.16. Sa structure se compose de deux couches 1^o le `Module Adaptor` et 2^o les `Chanel Adaptor`s séparées par une troisième l'`Internal Bus`. Une justification de cette structure est apportée par le prochain paragraphe, tandis que le paragraphe suivant en présentera les responsabilités fonctionnelles.
La structure des CC peut se résumer à une partie spécifique au processeur, se dénommant adaptateur de module (ou « **Module Adaptor (MA)** ») et une partie spécifique aux communications, qui se compose de sous-éléments, les adaptateurs de canal ou « **Chanel Adaptor (CA)** ». L'interconnexion de ces deux parties nécessite un troisième élément : le « Bus Interne » ou « **Internal Bus (IB)** ».
Cette structure est motivée par les cinq points suivants :

1. L'utilisation d'adaptateur d'interfaces standards telles que VCI[51, 38] devant pleinement répondre aux standards se révèle inadaptée de par la granularité grossière des fonctionnalités qu'ils implémentent :
   – les adaptateurs peuvent être des goulots d'étranglement pour les communications s'ils ne supportent pas des transferts évolués de données tels que des accès en rafale, permettant d'exploiter pleinement les bandes passantes des bus et processeurs ;
   – un surcoût en surface peut être engendré par l'usage d'adaptateurs dont seul un sous-ensemble de leurs capacités est réellement utilisé par l'association application-architecture.
2. Les besoins en communication des applications multiprocesseurs tendent à nécessiter l'utilisation de plusieurs ressources de communication concurrentes. L'utilisation de ponts classiques entre deux bus est pénalisant car en

nécessitant l'usage quasi-simultané des deux bus, ils restreignent le parallèlisme des communications. La possibilité de connecter les processeurs à plusieurs bus doit donc être offerte.

3. L'utilisation d'unités matérielles d'adaptation spécifiques à chaque connexion du processeur à un bus introduit un surcoût matériel. Ces unités comportent toutes une même fonctionnalité : le contrôle des échanges de données sur le bus natif du processeur. Or ce bus est déjà un goulot d'étranglement pour les communications puisque les transferts n'y sont pas concurrents, donc le partage de la fonctionnalité précédemment citée est envisageable avec une dégradation réduite des performances $\Rightarrow$ le MA.
4. L'insertion d'un bus entre le MA et le CA autorise l'usage partagé de ressources, entre plusieurs canaux, au sein du CC telles que de la mémoire enfouie ou un contrôleur de DMA.
5. La présence de ce bus interne réduit le nombre de composants différents nécessaires pour couvrir un espace de solution donné :
   – sans bus interne, il faut $\binom{N}{2} = \frac{N*(N-1)}{2}$ composants pour adapter 2 protocoles parmi N ;
   – avec bus interne, le même espace est couvert avec seulement $\binom{N}{1} + \binom{N}{1} = 2 * N$ composants.

Quelques inconvénients résident cependant dans cette solution. Le plus marquant de ceux-ci est l'usage d'un unique IB au sein du CC, conduisant à opérer un compromis global surcoût matériel/performances sur l'ensemble des canaux. Ce compromis peut alors se révéler localement (à un canal ne justifiant pas ces ressources) inadéquat.

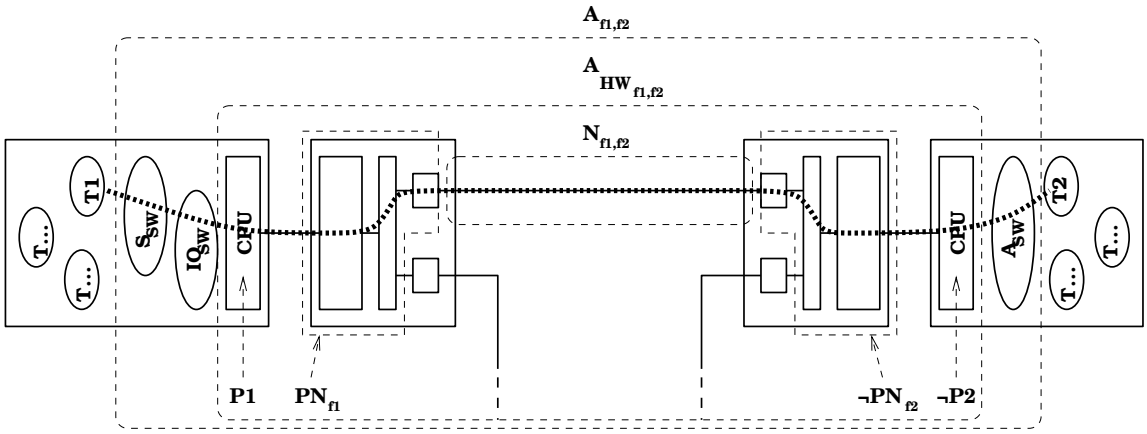


FIG. 2.17 – Cheminement d'une communication.

$$f_1 = A_{f_1,f_2} \circ f_2 \tag{2.1}$$

$$A_{f_1,f_2} = A_{SW_{f_1}} \circ A_{HW_{f_1,f_2}} \circ \neg A_{SW_{f_2}} \tag{2.2}$$

$$\begin{aligned} A_{SW_{f_x}} &= S_{SW_{f_x}} \circ IO_{SW_{f_x}} \\ \neg A_{SW_{f_x}} &= \neg IO_{SW_{f_x}} \circ \neg S_{SW_{f_x}} \end{aligned} \tag{2.3}$$

$$A_{HW_{f_1,f_2}} = P_1 \circ PN_{f_1} \circ N_{f_1,f_2} \circ \neg PN_{f_2} \circ \neg P_2 \tag{2.4}$$

$$\begin{aligned} PN_{f_1} &= MA_1 \circ IB_1 \circ CA_{f_1} \\ \neg PN_{f_2} &= \neg CA_{f_2} \circ \neg IB_2 \circ MA_2 \end{aligned} \tag{2.5}$$

FIG. 2.18 – Décomposition de la communication

Un CC doit répondre au problème suivant : soient T_1 et T_2 deux tâches concurrentes. T_1 requiert un service $f_1(data)$ que T_2 peut fournir par l'offre de son service $f_2(data')$. Une traduction de l'appel à $f_1()$ local à T_1 en une exécution de $f_2()$ déportée sur T_2 doit être apportée. Mais cette traduction est assujettie de contraintes fonctionnelles (la traduction doit conserver la sémantique de $f_1()$), de performances (les latences et coûts induits par l'implémentation de cette traduction peuvent être spécifiés) et architecturales (la structure de l'implémentation peut être contrainte par un motif architectural de référence et par des protocoles imposés).
Dans le contexte de l'architecture cible, cette problématique peut se formuler comme suit : soient les tâches T_1 et T_2 localisées sur des processeurs distincts de notre architecture cible telle qu'elle est dépeinte par la figure 2.17.

L'équation 2.1 de la figure 2.18 exprime la traduction de l'appel à f_1 en un appel à f_2. L'opérateur de traduction A_{f_1,f_2} peut grossièrement être décomposé en trois fonctionnalités (expression 2.2) :

1. l'adaptation logicielle entre la tâche T_1 et le matériel : $A_{SW_{f_1}}$;
2. la fonctionnalité du matériel reliant les deux tâches : $A_{HW_{f_1,f_2}}$;
3. l'adaptation entre le matériel et la tâche T_2 par $\neg A_{SW_{f_2}}$.

La partie d'adaptation logicielle abordée dans notre flot par [29] peut être décomposée (cf : 2.3) en une partie générique et portable : $S_{SW_{f_x}}$, et une partie spécifique à l'architecture : $IO_{SW_{f_x}}$. De même, la modélisation architecturale adoptée dans ces travaux permet de diviser la fonctionnalité d'adaptation matérielle en trois sous-familles comme l'illustre l'équation formelle 2.3 :

1. la frontière entre le domaine logiciel et le domaine matériel que sont les processeurs P_x et $\neg P_x$. Cette sous-famille assume des fonctionnalités figées qui sont :
   (a) la génération de signaux de niveau RTL à partir de l'exécution d'instruction ;
   (b) la réaction du flot d'exécution des instructions en fonction d'événements externes, c-à-d. la fonctionnalité complémentaire de la précédente.
2. le réseau d'interconnexions permettant de router, transporter et pourquoi pas transformer l'appel à f_2 : N_{f_1,f_2} ;
3. une couche permettant de traduire, contrôler et synchroniser les échanges entre les processeurs et le réseau d'interconnexions : PN_{f_1} et $\neg PN_{f_2}$.

La structure des adaptateurs matériels de communication permet de décomposer PN_{f_x} en trois couches : 1^o le MA, 2^o le bus interne et 3^o le CA. Cette division apparaît dans l'équation 2.5. La fonctionnalité du premier (c-à-d. MA_x) prend en charge la traduction et la synchronisation entre les protocoles du bus interne et du processeur. Le bus interne assume l'aiguillage ou la concentration des requêtes ou offres de services entre le processeur et le réseau d'interconnexion. Quant aux CA ils doivent traduire les protocoles du bus interne et du réseau d'interconnexion, mais aussi gérer la synchronisation des terminaisons de la communication.
L'assignation de ces fonctionnalités aux différentes unités structurelles (MA, CA et IB) trouve les trois motivations suivantes :

1. cette analyse fonctionnelle n'adresse qu'une communication point-à-point entre deux tâches. Pour plusieurs communications certaines fonctionnalités qui ne sont pas spécifiques à celles-ci, vont pouvoir être partagées. Dans les formulations précédentes, il s'agit de toutes les fonctionnalités non marquées par les indices « $_{f_1,f_2}$ », « $_{f_1}$ » et « $_{f_2}$ ». Parmi ces fonctionnalités se trouvent :
   – P_1 et $\neg P_2$ implémentées chacune par un processeur de logiciel qui aura sûrement en charge l'exécution de plusieurs tâches et plusieurs communications ;
   – MA_x et IB_x qui sont des goulots fonctionnelles justement utilisés pour la concentration/déconcentration des communications au travers de l'unique port logique de communication du processeur.
2. le choix de ces fonctionnalités et de leurs implémentations doit absolument être modulaire afin de maîtriser la complexité inhérente à un nombre élevée de processeurs et de ressources de communication diverses. Des décisions locales à une terminaison doivent influencer au minimum les choix relatifs aux autres terminaisons de la communication ;
3. la possibilité de migrer la nature de fonctionnalités entre une implémentation logicielle et une implémentation matérielle doit être maintenue. En effet, le choix de réaliser des fonctionnalités soit dans la couche $IO_{SW_{f_x}}$, soit dans CA_{f_x} permet d'explorer des solutions d'implémentation selon les critères de coûts et de performances usuels.

Les coprocesseurs de communication ont pour modèles d'implémentation une spécification structurelle contenant une instance d'adaptateur de module et plusieurs instances d'adaptateur de canal.

L'adaptateur de module Le MA est une première couche d'adaptation en charge de traduire le protocole natif du bus du processeur en un protocole unifié pour le nœud de calcul, c'est-à-dire le protocole du bus interne. Un adaptateur de module est donc décomposable en trois sous-entités fonctionnelles.
La première est un adaptateur de protocole et concentrateur d'accès (`Translator FSM` dans la figure 2.19). Ce module doit adapter tous les accès provenant du bus natif du processeur en accès au bus interne en profitant au mieux

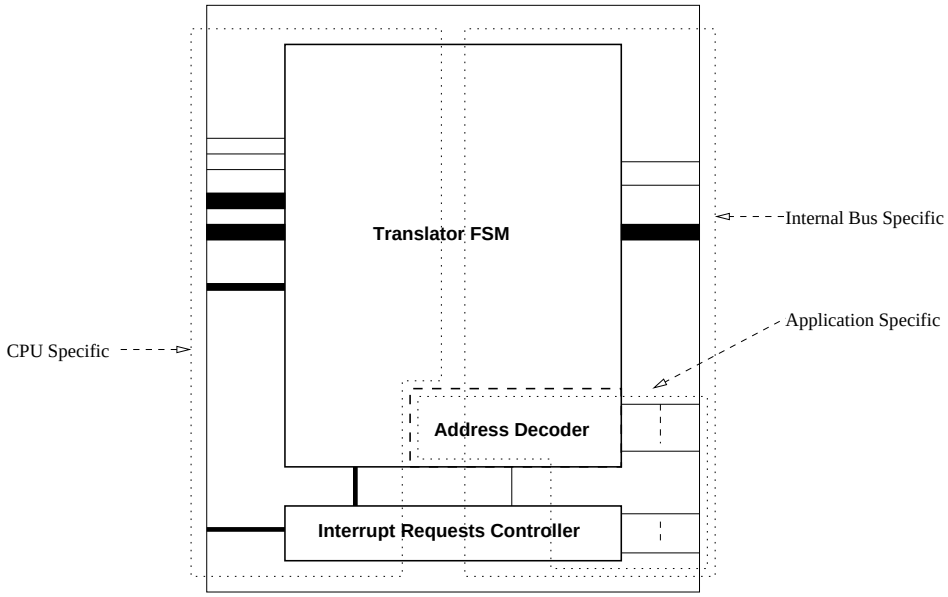


FIG. 2.19 – Adaptateur de module.

des possibilités du bus interne (Pipelining, Rafale). Il s'agit d'un cas particulier de CA. En plus de cette adaptation, il prend en charge la concentration des multiples accès concurrents en provenance des réseaux de communication en un unique accès séquentiel sur le bus mémoire de l'AL.
Un deuxième bloc est en charge de la gestion des requêtes d'interruptions (`Interrupt Request Controller` dans la figure 2.19). Tous les adaptateurs de canaux sont potentiellement des générateurs de requêtes d'interruptions concurrents. Or un cœur de processeur ne dispose que d'un nombre très restreint d'entrées pour les requêtes d'interruption. Il faut donc enrichir le mécanisme usuel d'interruption des processeurs avec une étape de concentration et de vecteurisation supplémentaire.
Un dernier bloc est responsable du routage par décodage d'adresse (`Address Decoder`) des accès du processeur vers le CA ciblé. Lorsque le coprocesseur de communication ne requiert pas l'utilisation de contrôleur de DMA interne, l'adaptateur de module se trouve être le seul maître du bus interne. L'arbitre de ce dernier se contente de décoder les adresses et de sélectionner l'esclave visé (un adaptateur de canal). Afin d'optimiser l'implémentation, cette fonctionnalité est embarquée dans l'adaptateur de module.
Les dépendances de l'adaptateur de module sont mises en évidence par des polygones en pointillés dans la figure 2.19. Trois types de dépendances sont distinguées :

1. Spécifique au processeur. Cette dépendance au processeur est surtout relative à l'interface de celui-ci. Le rôle de l'adaptateur de module étant justement de traduire le protocole natif d'un processeur donné en un protocole générique.
2. Spécifique au bus interne. Le choix du bus interne n'est pas unique. Suivant la nature des composants qu'il connecte, sa complexité peut varier du simple bus de données associé à des signaux de sélection, jusqu'au bus partagé multimaître comportant en sus des signaux de contrôle autorisant l'arbitrage des accès.
3. Spécifique à l'application. L'adaptateur de module est dépendant de l'application par les décodages d'adresses dont il a la charge :
   – Reconnaissance des accès au gestionnaire d'interruption.
   – Génération des signaux de sélection lorsque le bus interne est réduit à sa plus simple expression (un unique bus de données).

L'adaptateur de module est implémenté par des machines d'états finies au niveau transfert de registres, exprimées en SYSTEMC et VHDL.

L'adaptateur de canal L'adaptateur de canal, ou « Chanel Adaptor (CA) », est responsable du contrôle du protocole externe[5]. Son comportement peut être de deux natures symétriques et complémentaires :

[5]dicté par les spécifi cations

1. Si l'adaptateur est utilisé pour l'implémentation d'un port virtuel émetteur, il doit récupérer les données présentées sur le bus interne, éventuellement les mémoriser, les préparer pour l'émission[6] et enfin les émettre en gérant les signaux de contrôle spécifiques au protocole. Tout au long de ces étapes, l'adaptateur peut informer le processeur sur l'avancement et l'état de la transmission par le biais d'un registre de status (assigné d'une adresse en mémoire) et l'émission de requêtes d'interruption.
2. Si l'adaptateur est utilisé pour l'implémentation d'un port virtuel récepteur, il est en charge de la réception des données provenant du réseau de communication, de la gestion du protocole spécifique à ce canal, puis il doit décoder les données, possiblement les mémoriser, les préparer et les tenir à la disposition du processeur. La progression de chacune de ces étapes ainsi que la disponibilité des données sont indiquées au processeur au travers du registre de status et de requêtes d'interruption.

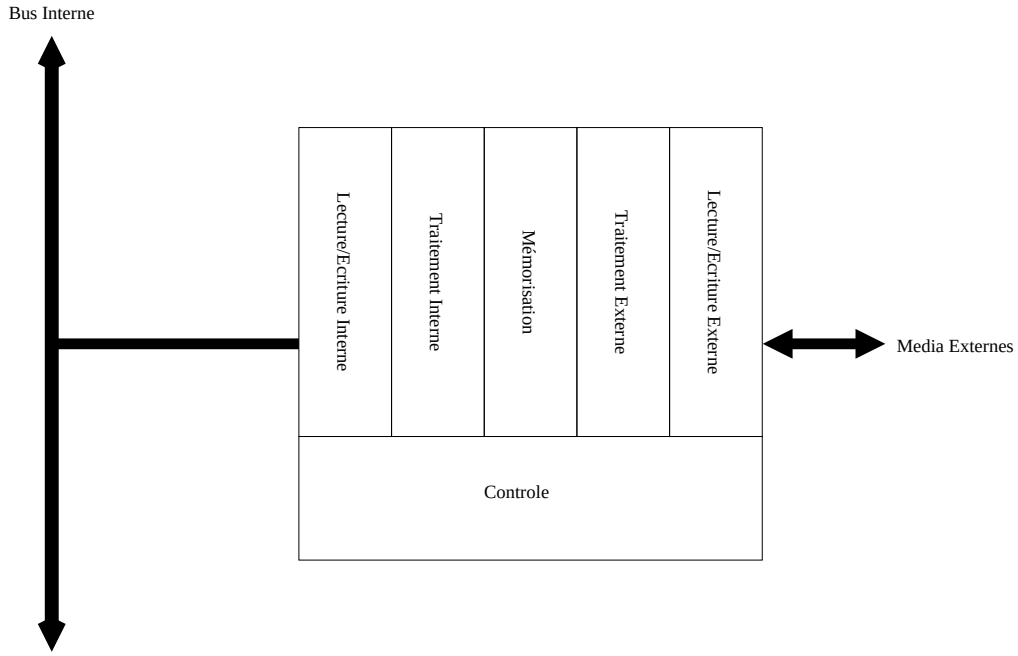


FIG. 2.20 – Décomposition fonctionnelle des adaptateurs de canal.

Six composantes fonctionnelles sont définies et illustrées par la figure 2.20, permettant ainsi de caractériser un adaptateur de canal. Il s'agit de :
– La couche de lecture et écriture interne permettant la capture et l'émission de données sur le bus interne. Cette couche peut être réemployée pour tous les adaptateurs de canaux supportant le même type de bus interne.
– La couche de traitement interne permettant un changement de représentation des données entre le bus interne et la mémorisation de celles-ci.
– La couche «Mémorisation» permet la mémorisation temporaire des données autorisant des communications entre producteurs et consommateurs désynchronisés.
– Les données sortantes (entrantes) doivent, avant d'être envoyées (mémorisée), être préparées, encodées, translatées[7] empaquetées (dépaquetées, décodées, translatées). Ces tâches incombent à la couche de traitement externe.
– La dernière couche latérale, dite de « Lecture/Écriture externe » prend en charge le contrôle du protocole externe et la présentation des données sur le canal.
– Ces cinq couches étant cascadées, une couche transversale est nécessaire afin de les synchroniser et les contrôler.

Suivant les spécifications de l'application, l'implémentation de ces différentes couches varie de la machine d'état finie au niveau transfert de registres, jusqu'à la simple connexion de signaux, en passant par des blocs de logique combinatoire. Les langages utilisés pour le codage de ces implémentations sont SYSTEMC et VHDL.

[6] Encodage, paquètisation, ...
[7] Traduction d'adresse entre le plan mémoire local et le plan global.

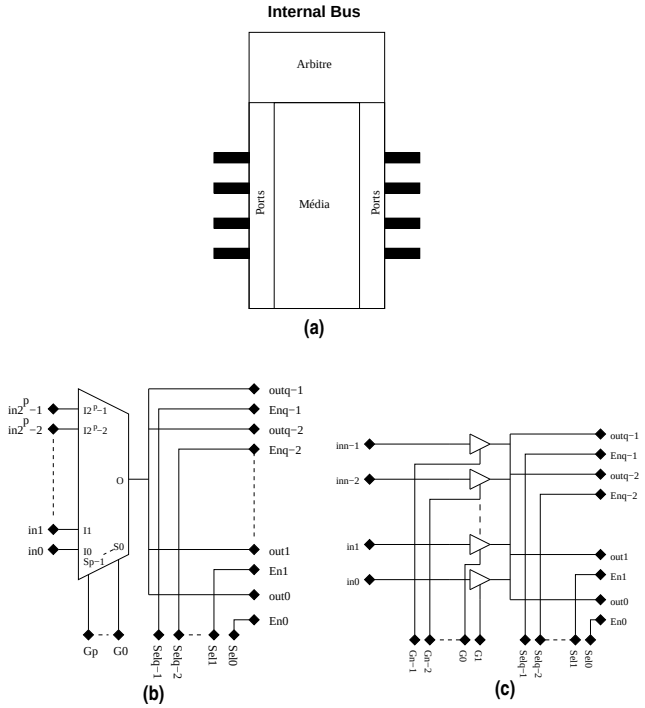


FIG. 2.21 – Bus Interne.

Le Bus Interne Ce bus interne interconnecte les différents adaptateurs de canaux à l'adaptateur de module. Mais là n'est pas sa plus grande contribution. Vu par les communications entrantes, il est le premier goulot d'étranglement tel une barrière au parallélisme. L'unicité du bus natif du processeur rend ce goulot inéluctable et donc l'utilisation de ce bus interne acceptable. De plus, plusieurs médias connectés au processeur peuvent nécessiter le partage d'une même fonctionnalité ou ressource (mémoire, encodeur, contrôleur de canal DMA, etc.). L'utilisation d'une version multimaître de ce bus interne permet alors d'amortir le coût de ces ressources en autorisant leur partage. Ce bus interne apparaît comme la colonne vertébrale du CC, aussi les efforts doivent être concentrés sur sa conception afin de le doter d'une bande passante et d'une police d'attribution pénalisant aussi faiblement que possible le débit des communications, tout en assurant un coût d'implémentation acceptable.

Conformément à la figure 2.21(a), le fonctionnement d'un bus interne peut être décomposé en trois parties : 1° La connexion des interlocuteurs au média au travers d'interfaces[8] pouvant être spécifiques à l'interlocuteur (Maître, Esclave, consommateur, producteur) ; 2° le média à proprement parler, en charge de véhiculer les données d'un producteur vers un ou plusieurs consommateurs ; 3° l'arbitre décidant de l'allocation du média et du routage pour établir une connection logique entre producteurs et consommateurs.

L'implémentation physique du média de communication peut être réalisée par l'utilisation de signaux partagés dont les valeurs sont résolues par l'utilisation de portes trois états, permettant de déconnecter des composants du bus, afin de ne laisser active qu'une unique sortie à un instant donné. Cette déconnexion est réalisée par l'insertion d'une haute impédance entre le bus et la sortie de la porte logique connectée. Cette insertion est possible grâce à une porte logique spécifique : la porte trois états («tri-state» pour les Anglo-Saxons), tel que l'illustre la figure 2.21(c). Pour y parvenir, il suffit de spécifier la valeur logique 'Z' (en HDL tels que SystemC et VHDL) pour les sorties attaquant le bus, lorsque ces dernières doivent être déconnectées. Mais il est aussi possible de ne router jusqu'au bus que la sortie intéressante à un instant donné. On peut utiliser pour cela, un multiplexeur comme la figure 2.21(b) le suggère. On peut aussi embarquer ce routage au sein du média par sa réalisation par des cellules de connections : les *micro-switch*es. Le bus *µNetwork*[86] et ses *Agent*s en est une excellente illustration.

Les *micro-switch*es étant matériellement la solution la plus coûteuse et la longueur des lignes ne justifiant pas leur usage, ils seront délaissés au profit des multiplexeurs et portes trois états sans pour autant en interdire l'utilisation. Le choix est donc restreint à l'utilisation de portes trois-états ou de multiplexeurs. La figure 2.22 donne les coûts surfaciques et performances temporelles comparés de ces deux options avec une technologie *AMS 3.20* ($0.35\ \mu m$) pour

[8] Ensemble de ports RTL

un bus large de huit bits et ayant 2^n connexions. Il est aisé de constater que les portes trois-états sont les moins intéressantes car leur utilisation comporte un coût surfacique linéaire relativement au nombre de connexions ainsi que des délais de connexion/déconnexion élevés dés que le nombre de connexions dépasse 16. Cependant, afin de démontrer la flexibilité de ce modèle architectural, l'utilisation de ces deux techniques est supportée.
Le contrôle des accès au média (arbitre ou décodeur d'adresse) en charge de générer les signaux de sélection est réalisé par des machines d'états finies configurables en termes de priorités d'accès ainsi qu'en adresses physiques assignées aux composants.

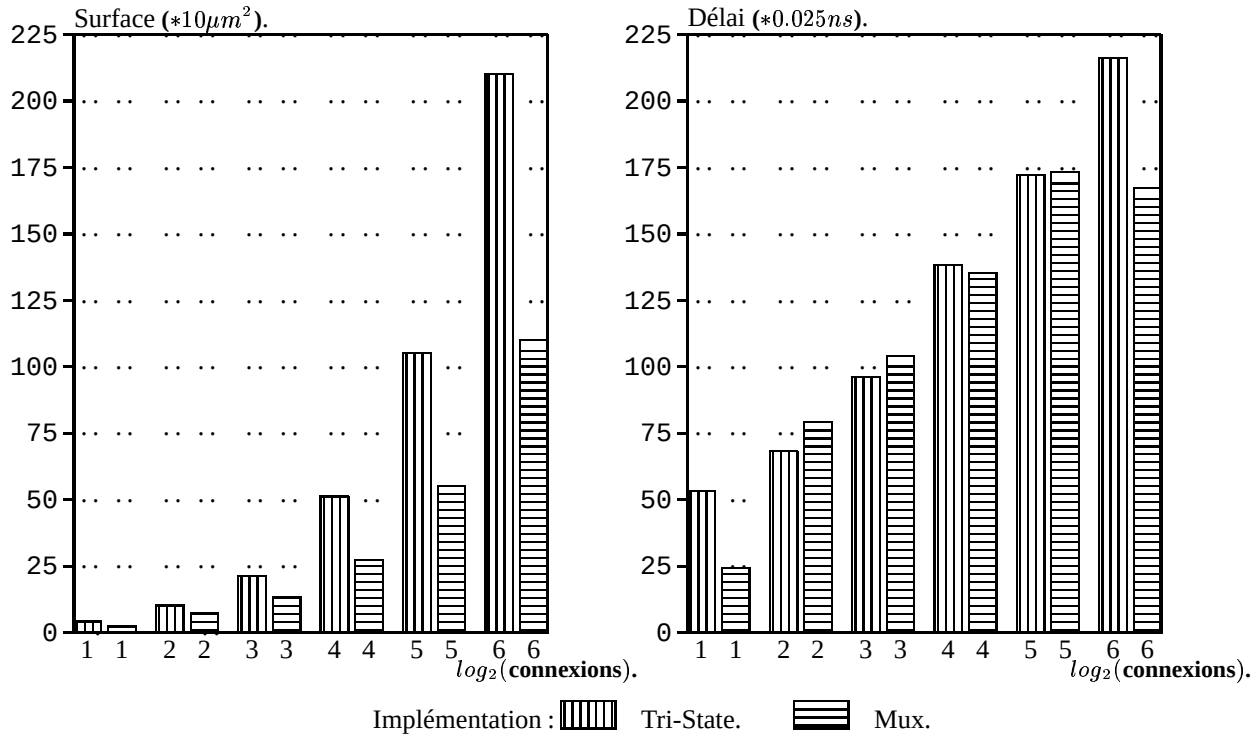


FIG. 2.22 – Comparaison des implémentations par multiplexeurs et portes trois états d'un bus 8 bits.

Les réseaux de communication

Au sein de notre modèle, les réseaux de communication sont appréhendés comme des composants prédéfinis ou IP. Bien que leur structure fonctionnelle soit généralement très régulière et donc facile à modéliser pour autoriser l'automatisation de leur conception, les fortes contraintes physiques qui y sont relatives (crosstalk, temps de propagation, intégrité du signal, ...) font que seuls des générateurs spécifiques et fortement conscients (dépendants) de la technologie sont aptes à remplir cette tâche. Pour cette raison, la modélisation de réseaux de communication apportée par ce modèle n'a pas pour but l'implémentation de ces-dits réseaux. Cependant, la validation globale d'une architecture étant aussi une cible de ce modèle, les réseaux de communication y sont représentés comme des modules matériels ayant des sous-blocs de contrôle (Arbitre, pont, ...) et des ressources de transmission (nets, chemin de données) pouvant être représentés à différent niveaux d'abstraction.

2.10 Conclusion

Ce chapitre a été consacré à la revue d'architectures matérielles mutliprocesseurs existantes afin d'en définir un modèle générique. La génération automatique de représentations de telles architectures pour leur implémentation est un objectif des travaux de cette thèse qui sera développé dans le chapitre 4.
Ce modèle générique est caractérisé par une structure multiprocesseur et une hétérogénéité des composants de traitement et de communication. Son organisation est elle aussi hétérogène. Les composants de calcul contrôle

indépendamment leurs flots de calcul et d'instruction. La topologie des ressources de communication ne garantie pas des accès uniformes des différents processeurs vers toutes les mémoires.
Ce modèle sera la base des modèles d'architectures générés par ces travaux.

Chapitre 3

Outils et modèles pour le raffinement automatique d'architectures

Sommaire

« Le tort des anciennes théories est d'avoir presque toujours négligé de donner le pourquoi des principes posés, ce qui fait que beaucoups d'exellentes choses ont été abandonnées et reprises, pour être abandonnées de nouveau. »

– Lieutenant-Colonel GERHARDT.

Ce chapitre est dédié à l'étude de méthodes et d'outils d'aide à la conception de systèmes monopuces. Le modèle de représentation des architectures multiprocesseurs monopuces sera brièvement rappelé avant d'introduire les niveaux d'abstraction utilisés lors de leur conception. Une étude des méthodologies de conception et des outils les implémentant pourra alors être conduite. Une distinction entre les approches d'exploration d'architectures et les flots de conception de ces architectures sera opérée. C'est la différence de leurs objectifs qui motivent cette séparation. L'exploration d'architecture permet de définir macroscopiquement la composition et le dimensionnement des ressources matérielles d'une architecture tandis que leur conception consiste en la mise en œuvre d'un modèle d'implémentation très détaillé. Enfin l'accent sera porté sur le flot de conception développé par l'équipe SLS dans laquelle ces travaux trouvent leurs contributions.

3.1 Proposition d'un modèle d'architecture multiprocesseur pour système sur puce

Dans cette section, le modèle d'architectures multiprocesseurs défini en 2.9 est repris afin de préciser le cadre d'étude des outils et méthodes de conception.

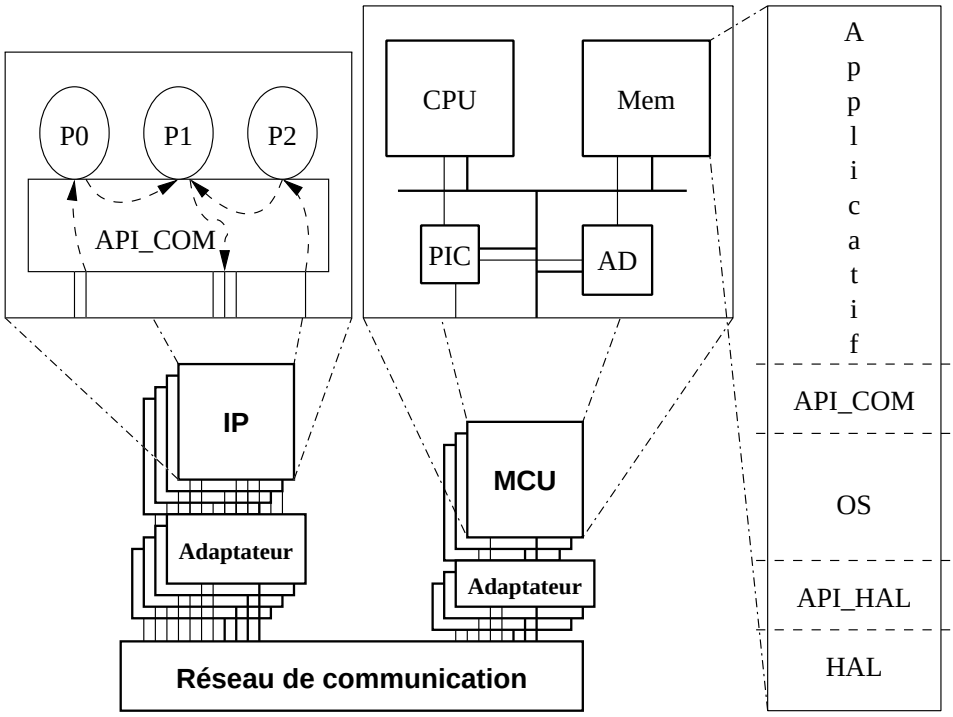


FIG. 3.1 – Modélisation d'une architecture multiprocesseur.

Ce modèle est représenté sur la figure 3.1 comme un assemblage de divers composants au dessus d'un réseau de communication. Ces composants sont des instances de processeurs matériels (`IP` sur la figure) ou des processeurs logiciels (`MCU`). Le découplage de la conception de ces composants avec la conception du réseau de communication peut conduire à des différences de protocoles que seul un adaptateur matériel peut résoudre. Le modèle d'exécution d'une application sur une telle architecture est structuré en couches fonctionnelles d'implémentations logicielles ou matérielles. Le modèle de la figure 3.1 présente les déclinaisons de cette structure sur les deux types de ressources de calcul mises en œuvre : 1^o les processeurs matériels (`IP`) et 2^o les processeurs logiciels (`MCU`).
L'utilisation d'un processeur logiciel générique ne permet un cycle de conception rapide uniquement que si on lui associe des couches permettant d'abstraire l'architecture matérielle à la vue du code logiciel applicatif. On distingue ainsi, sur la figure 3.1 quatre couches permettant cette abstraction. Ce sont :

1. Une couche implémentant en logiciel des services non fournis par l'architecture matérielle. Elle est désignée par le terme système d'exploitation ou Operating System (OS) pour les anglo-saxons. Les services implémentés peuvent permettre le partage d'une même ressource pour l'exécution de plusieurs entités logicielles.

2. Les services de communication utilisables par le code applicatif sont modélisés par un ensembles d'interfaces. Ces interfaces ou API_COM permettent d'unifier/standardiser l'usage des ressources de communication quelles qu'elles soient.
3. La conception d'un système d'exploitation est généralement une tâche ardue et focalisée sur une architecture. La couche API_HAL offre un découplage des fonctionnalités de l'OS des ressources matérielles (le processeur, les périphériques). Ainsi un système d'exploitation peut être porté à moindres frais sur une nouvelle architecture.
4. L'API_HAL ne définit qu'un ensemble de services et leurs usages. Le fossé séparant la fonctionnalité de ces services des ressources les implémentant (processeur, périphériques) nécessite une couche d'adaptation logicielle : la Hardware Abstraction Layer (HAL).

Ce besoin de découpler la fonctionnalité des communications est aussi valable pour les composants de calcul matériels. Or ces derniers se distinguent des processeurs logiciels par une flexibilité plus limitée. Ce découplage est alors localisé dans l'adaptateur de communication. Les fonctionnalités embarquées reposent là encore sur une interface de communication ou API_COM dont les services sont implémentés par l'adaptateur.

3.2 Flot de conception

Un flot de conception générique de système monopuce est composé de trois étapes principales ainsi que l'illustre la figure 3.2. On distingue en premier lieu une opération de spécification des fonctionnalités systèmes associées à des exigences non-fonctionnelles telles que des besoins de performances et des limitations de coûts. Puis une procédure d'exploration architecturale est entreprise afin d'arrêter les caractéristiques fondamentales de l'architecture (telles que les nombres et types des ressources de calcul, de mémorisation et de communication). Ce modèle macroscopique de l'architecture (la « macro-architecture ») alors défini sert de carcan à l'étape de conception où les fonctionnalités de l'application sont raffinées jusqu'à permettre leur implémentation sur silicium. Parmi ces services se trouvent les fonctionnalités de communication dont l'implémentation au travers d'adaptateurs matériels autorisant la connexion des composants de calcul aux ressources de communication est une contribution de ces travaux.

3.2.1 Abstraction du logiciel

La branche de conception des fonctionnalités assignées à une implémentation logicielle (à gauche dans la figure 3.2) permet d'obtenir un modèle exécutable sur un processeur donné de ces fonctionnalités. Cette démarche repose sur le raffinement successif partant des spécifications systèmes utilisant des modèles en Very-High-Level Language (VHLL) tel que UML-StateChart. Une étape de recodage des fonctionnalités, possiblement accélérée par l'utilisation de générateur automatique, est opérée afin de 1º tenir compte de la nouvelle topologie des communications héritée du *Mapping* (partitionnement) et de 2º disposer de descriptions compatibles aux flots de développement relatifs à chaque processeur embarqué. Le code obtenu est généralement modélisé grâce à un High-Level Language (HLL) tel que les langages C, C++, ou Java. La propriété de ces langages étant de ne pas être spécifiques à une architecture, des appels aux fonctionnalités d'une couche logicielle d'adaptation, la Hardware Abstraction Layer (HAL), permet d'accéder aux ressources spécifiques de l'architecture. A ce niveau de représentation le code applicatif en HLL est conservé et a été adjoint de code spécifique modélisable en C de bas niveau, voir en assembleur. Enfin, une étape traditionnelle de compilation et édition de liens permet d'obtenir le code exécutable dit de niveau Instruction Set Architecture (ISA).

3.2.2 Abstraction du matériel

La conception des processeurs matériels prend elle aussi, comme nous le montre la branche à droite de la figure 3.2, sa source dans les modèles VHLL. Ici encore une opération de recodage est requise pour répondre aux modifications structurelles du modèle de l'application et assurer la compatibilité aux flots d'implémentation de matériel existants. Le code résultant en Hardware Description Language (HDL) tel que Verilog ou VHDL décrit un algorithme de réalisation des fonctionnalités sans pour autant en spécifier les ressources nécessaires. Il est donc qualifié de l'adjectif « Comportemental ». Une étape de synthèse de haut niveau ou HLS permet d'allouer les ressources nécessaires, puis de les assigner à quantum de l'algorithme et enfin de réordonnancer leur exécution. La description résultante modélise l'implémentation des fonctionnalités au travers de l'évolution de leur état atteint après chaque événement de

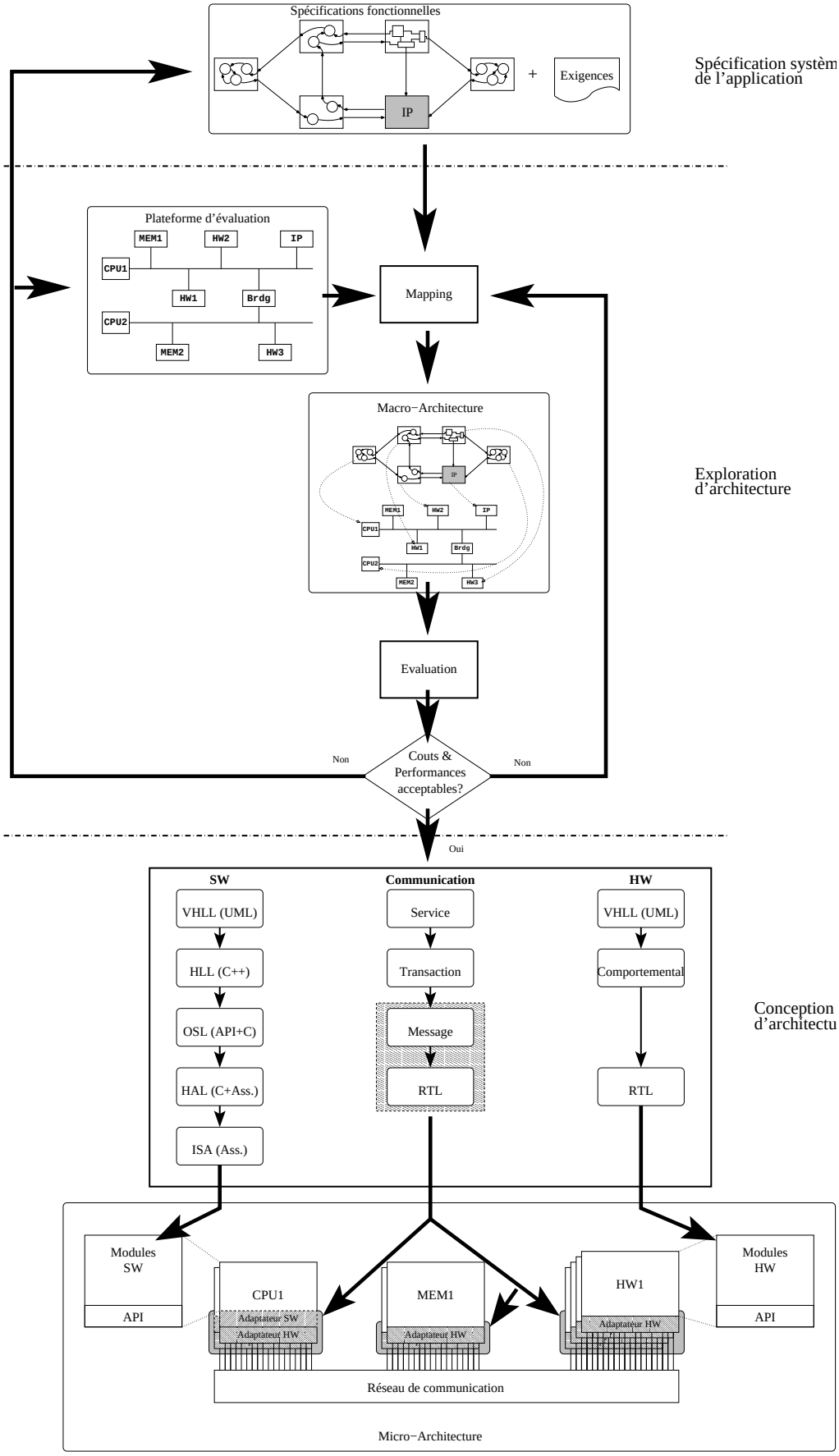


FIG. 3.2 – Etapes d'un flot générique de conception de systèmes monopuces

séquencement. Le niveau de cette description est donc le Register Transfert Level (RTL) qui est bien connu pour son aptitude à l'implémentation jusqu'au silicium.

3.2.3 Abstraction de la communication

La colonne centrale de l'étape de conception dans la figure 3.2 présente les niveaux *Service*, *Transaction*, *Message* et *Micro-architecture*. Ces quatre niveaux définis par [64], sont introduits dans les quatre paragraphes suivants.

Le niveau *Service*

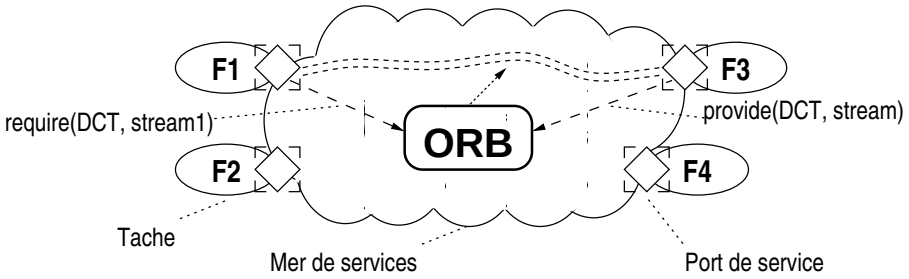


FIG. 3.3 – Spécification au niveau *Service*.

À ce niveau d'abstraction, le comportement est décrit par un ensemble de tâches fournissant et utilisant des services. Les communications sont des appels et des réponses d'offres de services. Elles sont décrites avec la couche *Application* de OSI. Le réseau de communication n'est pas structuré et est appelé *Mer de services*. Les tâches sont connectées à celle-ci par un unique port : le *port de service*. Ces ports peuvent émettre sur la *Mer de service* des requêtes ou des offres de services. Celles-ci sont collectées par une unité de contrôle : l'*Object Request Broker (ORB)*. Ce dernier peut assurer la connexion et le routage dynamique entre le client et le serveur de services.
La figure 3.3 donne un exemple d'un tel modèle. On peut y distinguer quatre tâches concurrentes : *F1*, *F2*, *F3* et *F4*. À un instant donné la tâche *F1* nécessite le service de calcul de DCT. Son *port de service* émet donc une requête pour obtenir ce service. L'ORB qui sait que la tâche *F3* peut fournir un tel service et qu'elle est disponible, établi dynamiquement un lien virtuel de communication entre *F1* et *F2*.
Une telle description est réalisable en utilisant le modèle Common Object Request Broker Architecture (CORBA)[23].

Le niveau *Transaction*

Le système est décrit sous la forme de réseau de nœuds de calcul concurrents et communiquant par l'échange de transactions encapsulant les données. Le comportement est donc décrit par des regroupement de tâches concurrentes inter-communicantes via une topologie explicite de média capable de transporter des données de type générique. Les langages *Unified Modeling Language (UML)*[78], *System Description Language (SDL)*[8] et *StateCharts*[34] sont de bons candidats pour de telles modélisations.
La figure 3.4 dépeint une telle description mettant en œuvre deux modules (M1 et M2) encapsulant chacun plusieurs tâches (F1, F2, F3 et F4) concurrentes. Une communication est émise à l'aide de la primitive *send* permettant le routage implicite d'une donnée vers la tâche réceptrice. Ainsi *F1* peut émettre des données en direction de *F3* par `send(F3,data1)`. La réception de communication est prise en charge par l'exécution de la primitive *get*. *F3* pourra donc récupérer dans sa variable locale *y* les données véhiculées par tous les messages la ciblant grâce à `y = get(data3)`.

Le niveau *Message*

Le niveau **Message** permet la description de notre système électronique sous forme de nœuds de calcul et de liens de communication aux ressources allouées[1], sans pour autant se préoccuper des détails de leurs implémentations en terme de structure, de signaux de communication et d'algorithmes détaillés de calcul. Le modèle du système hérite donc des

[1] lors d'une étape préalable de partitionnement.

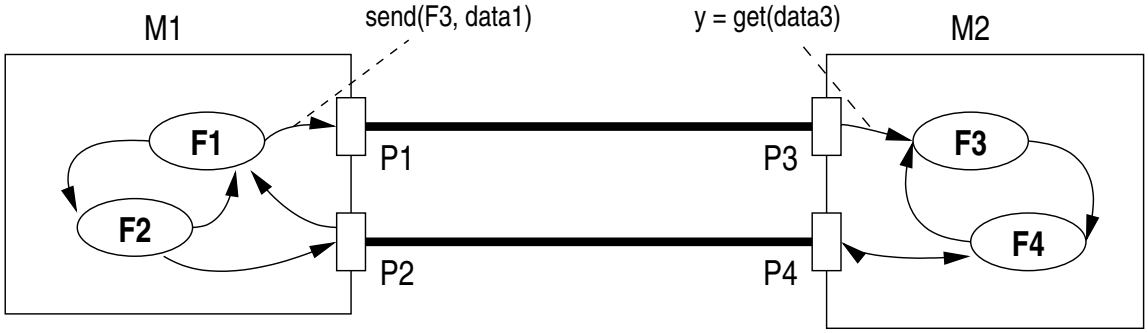


FIG. 3.4 – Spécification au niveau *Transaction*.

caractéristiques (protocoles de communication, performances) relatives à chacune de ces ressources, permettant ainsi une validation des choix architecturaux par des simulations et des analyses plus précises.
La figure 3.5 présente un exemple d'un tel modèle. Ici les tâches internes à un même module ne sont concurrentes que si les ressources disponibles le permettent. Dans le cas contraire il est requis de procéder à leur ré-ordonnancement soit statique si cet ensemble de tâches peut être modélisé par un unique graphe de dépendance de tâches, soit dynamique si ce graphe ne peut être construit. Ainsi les tâches *F1* et *F2* doivent se partager la même ressource de calcul *M1* (un unique CPU schématiquement identifié par *SW*) et leur exécution sera gérée par un ordonnanceur logiciel tel que présenté dans [73, 95]. Les communications sont gérées par les primitives Application Program Interface (API) telles que *write* et *read*. *F1* peut envoyer des données par `write(P1,data)` à *F3* qui les recevra grâce à `read(d,P2)`.

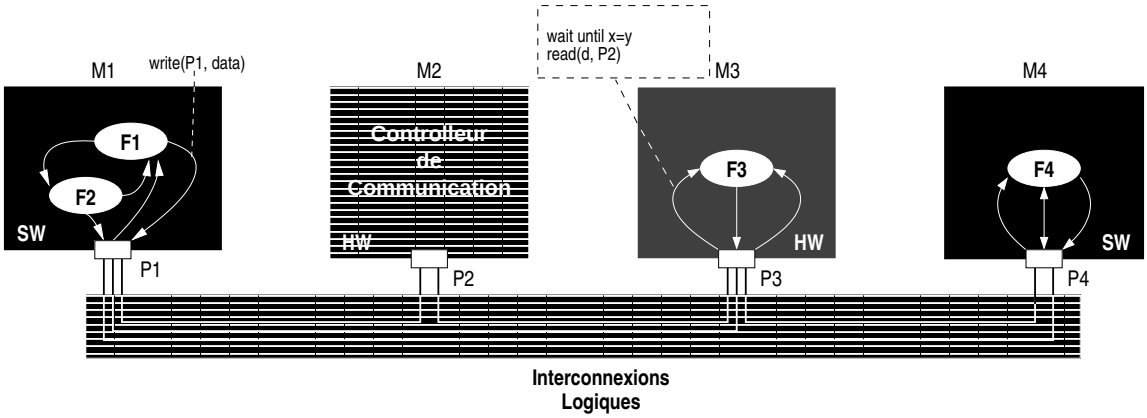


FIG. 3.5 – Spécification au niveau *Message*.

Une architecture dont les composantes sont décrites au niveau *Message* est dite « **Macro-Architecture** ». Cette dénomination provient de la vue macroscopique des unités de calcul, de communication et de mémorisation qu'offre une telle architecture.

Le niveau *Micro-architecture*

À ce niveau d'abstraction, les composantes matérielles sont modélisées au niveau *RTL*, tandis que les fonctionnalités logicielles sont représentée par du *Code objet*. Les communications sont maintenant détaillées jusqu'au travers de la couche *Liaison de données*.
Un exemple de modélisation *Micro-Architecture* est présenté dans la figure 3.6, la présence des composants suivants est notable :
– Un système d'exploitation (*OS*). Il a pour charge d'abstraire l'architecture matérielle au près du code applicatif :
  – par l'attribution temporaire des ressources (processeur ou autres) à une tâche (exemple : **M1**) ;
  – par l'implémentation de services de haut niveaux comme les communications (les couches les plus élevées du modèle OSI) et la synchronisation inter-tâches, la configuration et l'utilisation d'accélérateurs, les communications inter-processeurs (**M1** $\leftrightarrow$ **M2**).

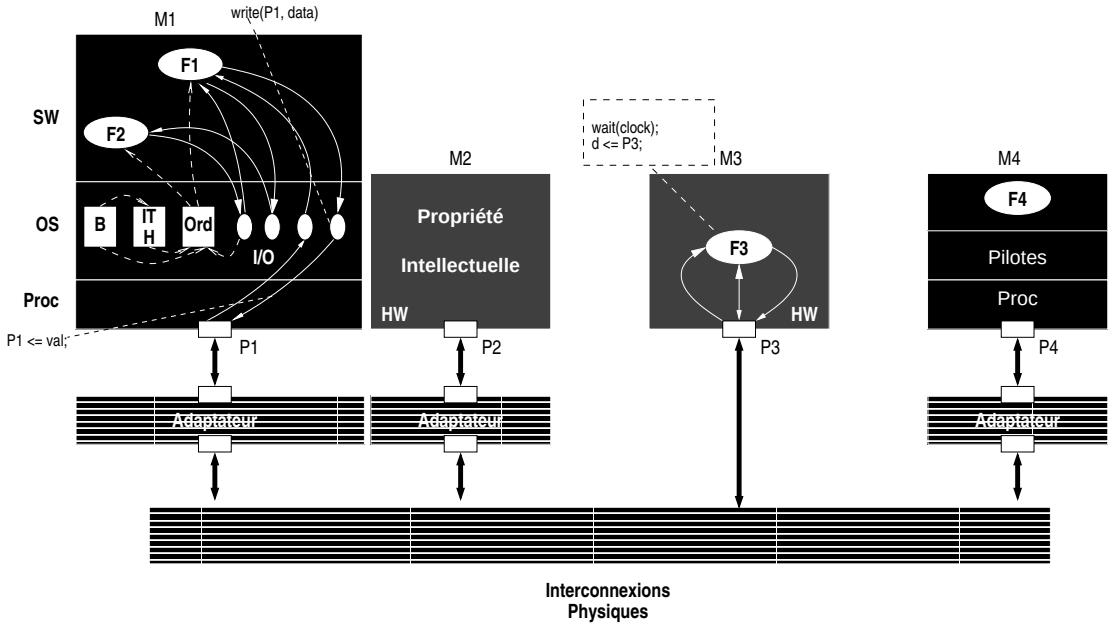


FIG. 3.6 – Spécification au niveau *Micro-Architecture*.

– Des adaptateurs matériels de communication permettant l'adaptation du calcul exécuté sur le processeur aux médias de communication.
– Une implémentation physique des médias de communication comportant les diverses unités de routage et de contrôle.

La transposition de la « Macro-Architecture » en une « Micro-Architecture » et plus particulièrement la génération des modèles RTL des adaptateurs matériels de communication est la contribution prépondérante de ces travaux.

3.3 Outils et méthodologies de conception d'architectures

Dans cette section une représentation générique du flot de conception de système monopuce sera introduite (sous-section 3.3.1) afin d'illustrer le contexte de l'utilisation des travaux de cette thèse. Puis des outils aidant les concepteurs de ces systèmes lors des décisions architecturales seront étudiés dans la sous-section 3.3.2. Enfin, des approches affairantes au même domaine d'application seront introduites dans les sous-sections 3.3.3 et 3.3.4.

3.3.1 Flot générique de conception

Le flot de conception (figure 3.7) d'un système monopuce est essentiellement basé sur deux étapes distinctes : 1° l'exploration d'architecture et 2° l'implémentation ou réalisation d'architecture. Les choix décisif de types de composants, de leur nombre ainsi que leur respect des performances spécifiées sont faits lors de l'étape d'exploration. Il en résulte un modèle de l'application que l'on appellera « architecture de référence » servant de référence et de spécification à l'étape de réalisation.
Peu de détails sont ici nécessaire à sa représentation. Elle peut ainsi être exprimée au travers d'une macro-architecture. La dernière étape, ou étape de réalisation est la plus pragmatique. Elle se « contente » de générer des modèles exécutables/synthétisables de l'architecture en intégrant les différents composants spécifiés par « l'architecture de référence ». Le résultat de ce raffinement peut être modélisé par une micro-architecture.
Deux approches se distinguent :
– une première approche basée sur la génération d'une architecture spécifique pour répondre aux besoins de l'application ;

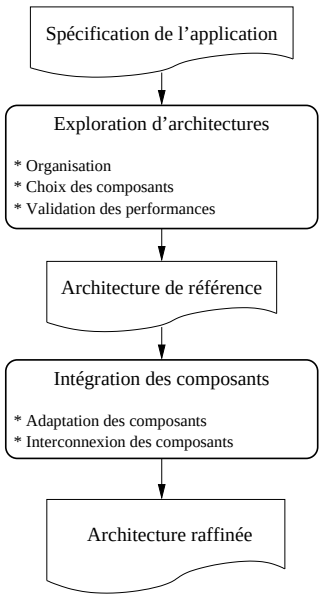


FIG. 3.7 – Flot générique de conception.

– une seconde articulée autour de l'assemblage de composants interconnectés à la périphérie d'un réseau de communication fixé et préalablement conçu. Ce réseau représente la véritable colonne vertébrale de l'architecture, puisque les performances en sont directement héritées.

3.3.2 Outils d'exploration d'architectures

L'exploration d'architecture consiste en l'élaboration de l'architecture la plus propice à répondre aux exigences de performances et aux contraintes de coûts (surface, consommation, etc...). Elle nécessite un grand nombre d'essais [5], chacun permettant de fournir des évaluations fidèles d'architectures diverses. Aussi la disponibilité d'un outil semi-automatisant cette exploration resterait fortement appréciée des architectes systèmes. De tels outils génèrent une architecture en partant d'un modèle fonctionnel de l'application et cherchent à répondre aux besoins de celle-ci tout en respectant un jeu de contraintes. Généralement, ils sont constitués d'un ensemble d'étapes permettant 1^o de fixer les composants et la structure de l'architecture et 2^o de déterminer l'organisation de cette dernière par l'assignation des fonctions de la description initiale de l'application aux composants de l'architecture. Un modèle générique d'un tel outil est représenté par la figure 3.8. On distingue trois types d'entrées admises par l'outil :

1. la spécification des fonctionnalités de l'application ;
2. les contraintes que doivent respecter la future implémentation de ces fonctionnalités ;
3. le modèle architectural utilisé pour composer et personnaliser cette implémentation.

L'architecture résultante est structurée en un ensemble de composants spécifiques à un traitement, et son organisation assure l'utilisation de ces composants de façon cohérente et convergente vers la fonctionnalité de l'application. Le flot est itératif et semi-automatique. Il contient une boucle permettant l'évaluation d'un modèle de performance de l'architecture afin de vérifier le respect des contraintes, puis de décider et de localiser des améliorations.
Quelques outils d'exploration parmi les plus représentatifs de ce domaine sont étudiés dans les paragraphes suivants. Ces outils sont *Cosyma* de l'université de Braunschweig, *Archimate* de *TNI-Valiosys*, *VCC* de *Cadence*. Les différences entre ces outils peuvent être qualifiées en fonction de :

1. la flexibilité des plateformes architecturales supportée. Leur définition peut être soit externe, ce qui amène une grande flexibilité, soit implicite car noyée dans les algorithmes de l'outil, limitant ainsi fortement l'espace de solution.
2. les axes de recherche qui permettent de réduire l'espace de solution en des sous espaces de tailles plus réduites. Cette capacité est directement reliée à la richesse des contraintes supportées par l'outil. Cependant, la diversité et la nature de ces contraintes telles que la *Testabilité*, la *Réutilisabilité* et la *Flexibilité*, rendent difficile la définition de métriques capables de guider l'exploration par la recherche de leurs extremums.
3. les niveaux d'abstraction pouvant être utilisés pour exprimer le modèle d'entrée de l'application. Ce critère reflète essentiellement le découplage entre l'expression des fonctionnalités de l'application et une quelconque implémentation, exprimant ainsi la liberté d'action de l'outil. Les différences suivantes doivent donc être mises en évidence :
   (a) l'abstraction du découpage des fonctionnalités et l'allocation – assignation de composants architecturaux ;
   (b) l'abstraction structurelle et protocolaire de la communication de l'allocation – affectation de réseaux ou bus de communication.
4. le niveau d'abstraction des modèles de sorties qui renseignent sur l'effort restant à apporter pour implémenter le système.
5. la méthode d'exploration de l'espace de solution qui peut être :
   – manuelle, l'outil n'apportant alors qu'une aide à la manipulation de modèle fastidieux ;
   – exhaustive et basée sur des algorithmes conduisant à une solution optimale sous contraintes ;
   – heuristique et conduisant à une solution acceptable, possiblement optimale.
6. la méthode d'évaluation est elle-aussi un critère de distinction. Elle peut-être :
   – statique et basée sur une analyse globale de l'assignation fonctionnalités $\leftrightarrow$ ressources de calculs et de communications ;
   – dynamique par la simulation/exécution d'un modèle de performance de cette assignation, puis une analyse à posteriori des réponse de ce modèle.

Une comparaison rapide de ces outils est donnée par le tableau 3.1.

	Cosyma	TNI Valiosys	VCC
Flexibibilité	Architecture fixe	Architecture générique Point-à-point	Plateforme définie par l'utilisateur
Axe d'optimisation	Surface/vitesse	Évaluation de la vitesse à posteriori.	
Modèle d'entrée	CDFG unique, pas de parallélisme	Modèle fonctionnel avec parallèlisme et communication abstraite	
Modèle de sortie	HW RTL et **1** SW statiquement ordonnancé	HW RTL et **n** SW statiquement ordonnancés	Modèle de performance de l'architecture
Méthode d'exploration	Partitionnement automatique avec architecture monoprocesseur	Assignation directe groupes de tâches fonctionnelles $\leftrightarrow$ unités matérielles	Partitionnement manuel sur plateforme prédéfinie.
Méthode d'évaluation	Analyse statique	Analyse dynamique par simulation de modèles de performances	
Remarque	Projet très ambitieux, mais la modélisation par un CDFG unique rend difficile la génération d'architectures efficaces	Le raffinement s'opère par simple transposition sans aucune optimisation. Les résultats dépendent grandement de la qualité du modèle d'entrée.	

TAB. 3.1 – Outils d'exploration d'architectures.

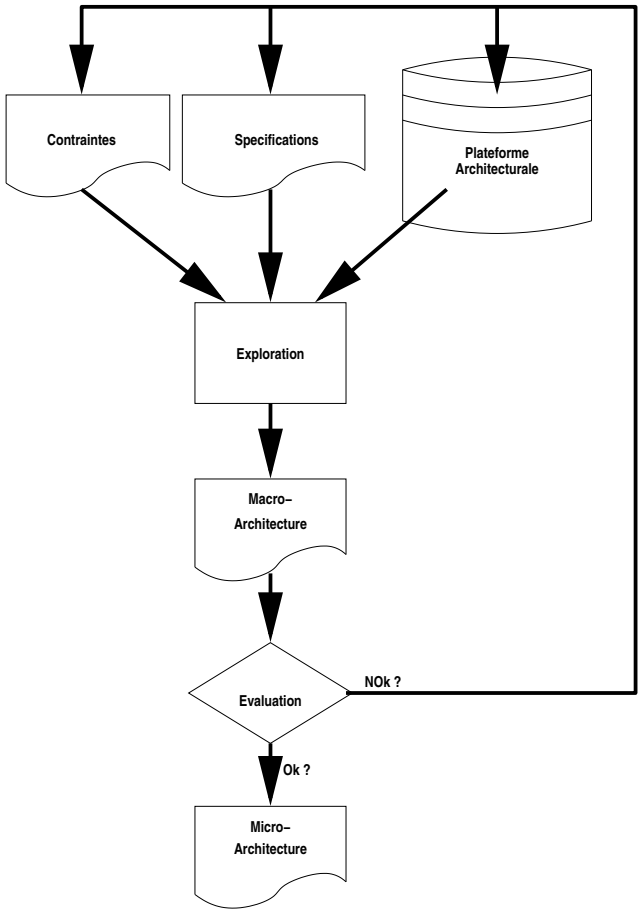


FIG. 3.8 – Modèle générique d'un outil d'exploration d'architecture.

Cosyma

COSYMA [110, 35, 9, 27] de l'université de Braunschweig est un environnement de co-conception permettant la mise en œuvre d'architecture limitée à un processeur logiciel RISC et plusieurs accélérateurs matériels. L'ensemble communique par des mémoires partagées. Le modèle d'entrée est ici un graphe unique de tâches, le Control & Data Flow Graph (CDFG). Le comportement de chacune est exprimé en langage C. Cosyma permet en plus de la génération d'une architecture spécifique, d'analyser les performances de cette dernière. Les sorties de cet outil sont d'une part le code binaire prêt à être exécuté sur le microprocesseur et d'autre part le code VHDL de niveau *RTL* décrivant le comportement des accélérateurs.

TNI-Valiosys Archimate

Archimate est un outil d'exploration d'architecture issu des travaux de recherche [104, 45]. Il permet la génération d'un modèle *RTL* de l'architecture matérielle ainsi que la génération de code logiciel en langage C destiné à être exécuté par un ou plusieurs processeurs. L'entrée de cet outil est une description structurelle de l'application en termes de modules concurrents inter-connectés. Ces modules peuvent être spécifiés à des niveaux d'abstractions différents et avec des langages différents[37] tels que C, Cossap, Matlab, HDL, SDL.
Les sorties de cet outil sont :

– Une description structurelle raffinée de l'architecture matérielle. Les canaux de communication sont implémentés par des liens et des protocoles point-à-point ;
– Une implémentation logicielle de parties de l'application en hiérarchie de machines d'états entrelaçant le comportement de plusieurs tâches (machine d'états feuilles). Ces machines d'états sont codées en langage C.
– Des modèles *RTL* en VHDL pour les implémentations matérielles de sous-systèmes de l'application.
– Des modèles *RTL* en VHDL et du code C pour les interfaces logiciel↔matériel.

Le partitionnement est implicite au modèle fonctionnel de l'application : chaque groupe de fonctionnalité se voit

assigné de manière bijective une unique ressource de calcul (processeur de logiciel ou processeur matériel). Les implémentations de la communication restent malheureusement trop simpliste pour autoriser l'utilisation de cet outil dans un but autre que l'exploration architecturale. En effet, on déplore l'absence d'intergiciel pour l'adaptation logicielle des communications, une topologie et des solutions architecturales figée.

Cadence VCC

Virtual Component Codesign (VCC)[15] de Cadence est un outil d'exploration d'architecture qui suit le flot générique de la figure 3.8. La plateforme architecturale est définie indépendamment de l'application. Le concepteur décrit les fonctionnalités de son application sous la forme de graphe de tâches et peut exprimer des contraintes de vitesse d'exécution. Il doit ensuite assigner manuellement ces fonctionnalités aux différents composants de l'architecture. L'outil lui fourni alors l'évaluation d'un modèle de performance dont l'acuité dépend directement de la précision des annotations de chaque composant constituant la plate-forme architecturale. Aux vues de ces résultats, le concepteur peut aisément identifier les composants (processeurs, périphériques, bus et mémoires) pénalisant l'ensemble de l'architecture et apporter les corrections nécessaires. Aucun modèle dédié à l'implémentation n'est disponible.

3.3.3 Outils de synthèse de réseaux de communication

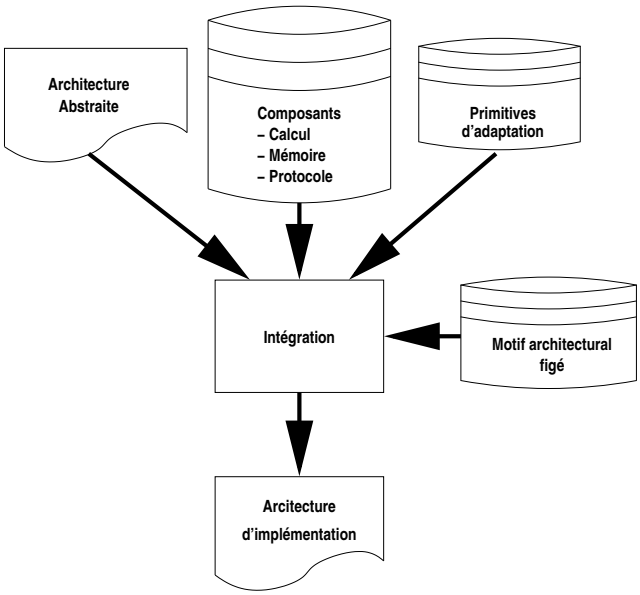


FIG. 3.9 – Modèle générique d'un outil d'intégration de composants avec synthèse du réseau de communication.

La génération d'architectures détaillées d'implémentation à partir du modèle de référence par les outils d'intégration de composant est représentée dans la figure 3.9. Les fonctions principales de ceux-ci sont :

1. le raffinement de l'architecture abstraite consiste en un remplacement systématique des modèles abstraits des composants de calcul et de mémorisation par les implémentations correspondantes. Ces modèles sont généralement constitués d'une instance d'un modèle réalisable du composant (processeur ou bloc mémoire) et d'un chapelet de composants périphériques nécessaire à son fonctionnement (mémoires locales, décodeurs d'adresses, bus et arbitres, contrôleurs de rafraîchissement ...). Cet ensemble constitue une Architecture Locale (AL).
2. la synthèse des réseaux de communication à partir de modèles virtuels (liens logiques ou composants abstraits) vers des modèles implémentables.
3. l'adaptation éventuelle des composants au réseau de communication par l'insertion entre ces deux entités d'adaptateurs matériels.

Parmi les outils disponibles, seuls *N2C*, *Program* et *Gaut* seront étudiés. Un récapitulatif des caractéristiques de ces outils, telles que les modèles de composants manipulés, la nature du réseau de communication, l'adaptation des composants et la génération d'un modèle du réseau d'interconnexion, est donné par le tableau 3.3.

CoWare Napkin-to-Chip

CoWare Napkin-to-Chip (N2C)[106, 105, 107] est un environnement de co-simulation et de génération d'interfaces. La méthodologie de *CoWare* s'articule autour d'un langage unique permettant la description de logiciel et de matériel à différents niveaux d'abstraction. Ce langage, le *CoWareC* est le fruit d'une extension de C, capable de modéliser la structure et le comportement d'une architecture à travers les quatre niveaux d'abstraction suivants :

– Le niveau *UnTimed (UT)* permettant une modélisation fonctionnelle d'applications, sans notion de temps. Les communications sont implicites, les ports sont manipulés comme des variables C volatiles. Ce niveau est donc équivalent au niveau *Transaction* introduit en 3.2.3.
– Le niveau *Bus Cycle Accurate (BCA)* ; dans ce niveau d'abstraction, le comportement des tâches est séquencé par une horloge, et contrôlé par un signal d'initialisation (reset). Ce niveau est équivalent au niveau *Micro-architecture* présenté en 3.2.3.
– Le niveau *Bus Cycle Accurate Shell (BCASH)* ; ce niveau apporte une enveloppe BCA aux modules d'abstraction UT, afin de les connecter à un environnement BCA.

Les applications sont modélisées dans ce langage en utilisant le modèle de calcul Remote Procedure Call (RPC) où ne subsistent que deux concepts d'exécution, les tâches maîtres et les tâches esclaves. Deux tâches n'appartenant pas au même module ne peuvent communiquer qu'au travers de leurs ports et d'un canal implicitement déclaré. Chaque tâche communiquant par un port se voit assigner une relation de maître ou (exclusif) d'esclave vis-à-vis de ce port. Elle peut soit :

– initier la communication, si elle est maîtresse du port et se geler en attendant la réponse en provenance de l'entité distante ;
– être activée uniquement lorsqu'une tâche maîtresse distante initie une communication débouchant sur le port esclave. Elle s'exécute alors complètement puis le port esclave envoie un acquittement implicite à sa contrepartie maîtresse.

Le réseau de communication est donc formé de connexions point-à-point maître-esclave.

Les architectures ciblées par N2C (figure 3.10) sont monoprocesseurs, les périphériques sont individuellement connectés au bus natif du processeur au travers d'instances d'adaptateurs (Bridge) dont les largeurs de données véhiculées et le nombre de cycles d'attentes sont statiquement configurables.

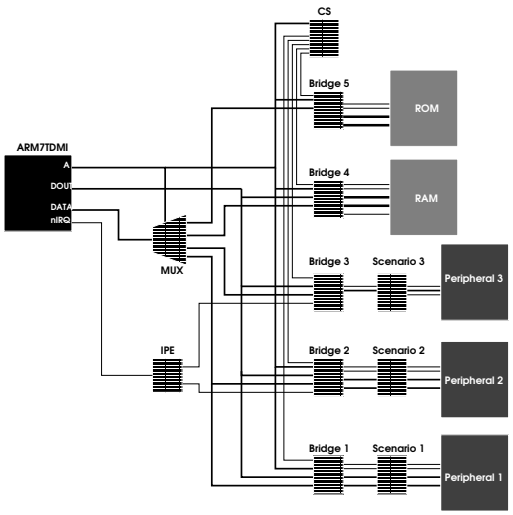


FIG. 3.10 – Architecture cible de Napkin-to-Chip

Les bibliothèques utilisées sont dénommées Processor Support Packages (PSP) et contiennent un modèle générique d'AL spécifique à un processeur et les modèles d'adaptateurs spécifiques aux protocoles. Ces protocoles sont implémentés par des connexions point-à-point physiques suffisantes pour satisfaire au modèle de calcul RPC adopté par CoWare. Le choix des protocoles supportés est le suivant :

NoHndshk : Utilisation d'un unique signal véhiculant les données. Aucun signal de contrôle, de synchronisation ou de sécurisation (parité) n'est utilisé. Les consommateurs de données échangées avec ce protocole doivent continuellement et périodiquement scruter leurs ports d'entrées.

Ce protocole voit donc son utilisation restreinte à la configuration de processeurs matériels par un processeur de logiciel.

EnHndshk : Ce protocole synchrone (partage d'un même signal d'horloge par les deux terminaisons du canal) utilise en plus du signal de donnée, un signal « *Enable* » autorisant sa prise en compte. Cette autorisation est matérialisée par l'affirmation de ce signal durant les cycles d'horloge voyant la donnée valide.

La donnée devant être prête lors de l'autorisation, ce protocole ne saurait être conseillé que pour les cas suivants :
- C'est le producteur de données qui initie la communication (« *outmaster* $\rightarrow$ *inslave* ») ;
- La latence de réponse du producteur esclave (« *outslave* $\rightarrow$ *inmaster* ») est fixe et connue.

FullHndshk : ou protocole *rendez-vous* à quatre phases permet la synchronisation des deux terminaisons à l'aide d'un signal d'initiation (« *request* ») et d'un signal d'acquittement (« *acknowledge* »).

MemEnHndshk : ce protocole est une évolution du protocole *EnHndshk* auquel un signal d'adresse a été ajouté. Il est ainsi possible de communiquer avec un processeur matériel comme avec un bloc de mémoire SRAM.

MemFullHndshk : l'ajout d'un signal d'adresse au protocole *FullHndshk* permet l'accès à une donnée dans un tableau enfoui dans le processeur matériel, tout en synchronisant cet échange par un mécanisme de *rendez-vous*.

Cette approche comporte plusieurs limitations. La restriction de modélisation du comportement au modèle de calcul RPC permettant à *N2C* la mise en place de l'ordonnancement de l'exécution des différentes fonctionnalités concurrentes sur des ressources restreintes (le processeur logiciel). Les communications sont restreintes au modèle point-à-point ; Les données échangées sont des vecteurs de bits (à tous les niveaux d'abstraction), ce qui suppose que les tâches applicatives doivent elles-même préparer et éventuellement découper les données en mots pouvant ëtre transmis. De plus ceci impose une représentation unifiée des données dès le niveau *UT* et restreint fortement les solutions d'implémentation des communications. L'absence d'un niveau d'abstraction équivalent au niveau « Message » (cf. 3.2.3) est implicitement traduite par une limitation de l'outil à ne supporter qu'un ensemble restreint d'architectures cibles, toutes issues du modèle générique monoprocesseur et à communications point-à-point.
L'absence de découplage entre les ressources de calcul (le processeur de logiciel) et le contrôle des communications ne permet l'exécution concurrente de fonctionnalités de calcul et des communications. L'utilisation de ressources allouées et assignées bijectivement aux canaux de communication interdit le partage de ressources matérielles (telles que des contrôleurs de DMA, d'encodeur, etc...) par plusieurs communications.

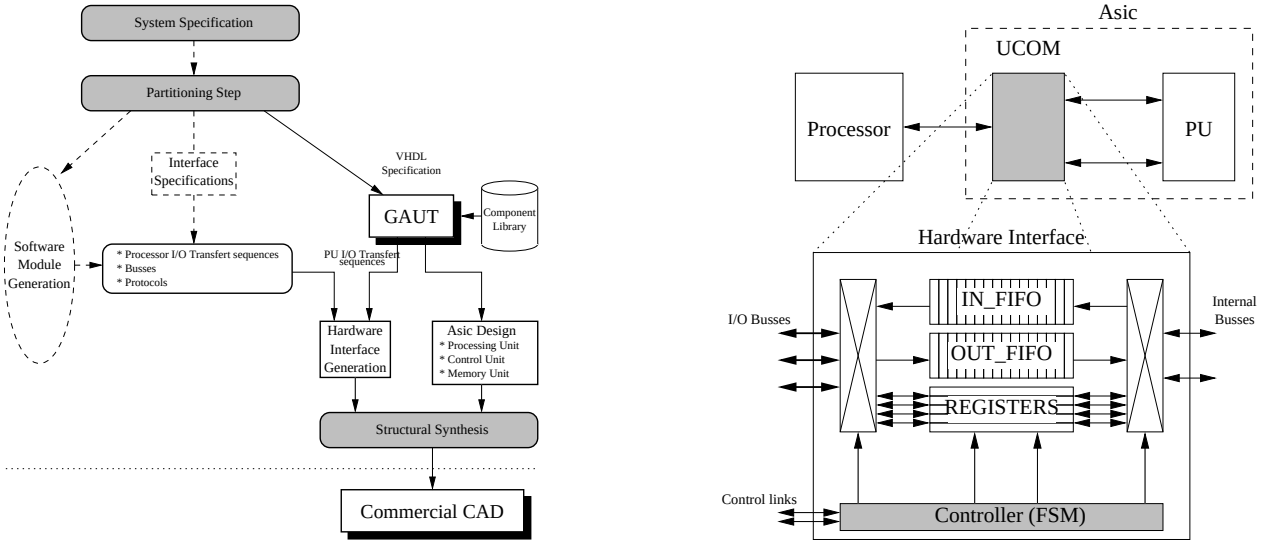
O'Nils et Program

[67, 68] présentent une méthodologie de conception d'architecture matérielle à base de processeurs communicants par des protocoles différents. Ces travaux traitent de la spécification et de la génération de ces adaptateurs de communications, mais cette opération n'est pas globalement automatisée.
Cependant un environnement de cosynthèse et de prototypage est proposé : [94]. Dans cet environnement, la génération d'adaptateurs matériels de composants est adressée par *Program*[65]. Conscient que les approches *Interface-Based* (telles que [79]) sont limitées par la disponibilité de bibliothèques suffisamment fournies pour adapter toute la diversité de protocoles existants, *Program* présente une nouvelle approche pour la spécification d'interface matérielle de communication, basée sur la modélisation du comportement de l'interface par une grammaire régulière. Un outil est alors utilisé pour générer un automate de reconnaissance des lexèmes, conceptuellement très proche des outils d'aide à la conception d'analyseur syntaxique pour les compilateurs tels que *GNU Yacc*. Il en résulte un modèle VHDL RTL, implémentant par des machines d'états finies, les automates de reconnaissance.
La contribution principale de ces travaux est une méthodologie et un outil de conception d'adaptateur basée sur une modélisation « grammaticale » du comportement de l'adaptateur, certes plus concise que par l'utilisation d'un HDL, mais encore manuelle.
Protocol Compiler[88, 80] de Synopsys est un autre exemple très similaire de cette approche basée sur des expressions régulières.

GAUT

GAUT[4] est un outil de synthèse comportementale (cf. figure 3.11(a)). Il permet la génération des interfaces matérielles appelées *Communication Unit (UCOM)*. Le réseau d'interconnexions mis en œuvre est une nappe de connexions point-à-point reliant un unique processeur de logiciel à plusieurs accélérateurs matériels. Il est basée sur

une analyse formelle des communications de l'application et sur leur assignation à des protocoles faisant partie des spécifications. Il permet de générer des modèles VHDL RTL synthétisable tant pour les accélérateurs que pour les unités d'adaptation. *Gaut* est orienté architecture monoprocesseur (cf. figure 3.11(b)). L'utilisation de cet outil est envisageable dans le cadre d'architectures multiprocesseurs, si son analyse globale des communications peut supporter la complexité des réseaux embarqués. De plus l'utilisation de connexions point-à-point rend coûteuse la mise à l'échelle du modèle architectural et donc son utilisation pour des systèmes multiprocesseurs fortement communicants.



(a) Flot de conception.

(b) Architecture cible.

FIG. 3.11 – Conception de haut-niveau avec GAUT.

3.3.4 Méthodologies et outils d'intégration de composants autour de réseaux de communication propriétaires

La diversité des réseaux de communication et de leurs implémentations rendent difficile le support d'un modèle générique par les outils d'intégration. Une solution consiste à restreindre les modèles de réseaux supportés à une famille de topologies et/ou de protocoles. Les méthodologies issues de cette démarche sont génériquement représentées par la figure 3.12. Leur principale spécificité réside dans la disponibilité d'un jeu figé (non extensible) de protocoles et de réseau de communication profondément accouplé à un modèle d'assemblage.

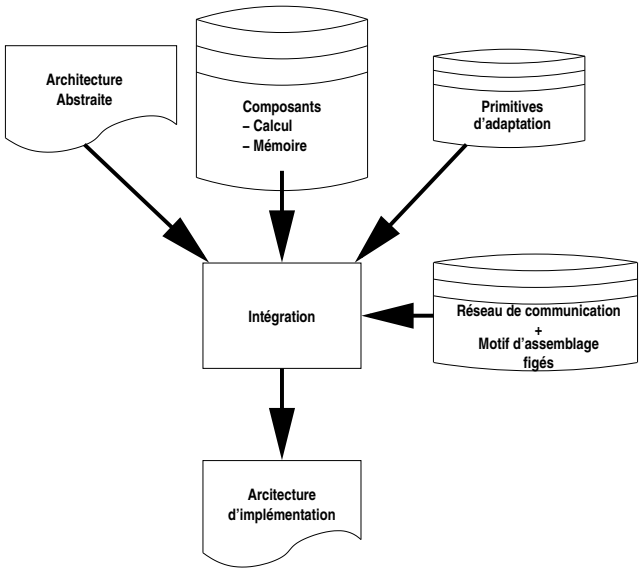


FIG. 3.12 – Modèle générique d'un outil d'intégration de composants avec réseau de communication fixé.

L'outil *Coral*, la solution *µNetwork* et les méthodes à base de standards d'interfaces sont présentés dans les paragraphes suivants. Un récapitulatif de leurs caractéristiques est formulé par le tableau 3.3.

IBM Coral

Coral[11, 10] permet de spécifier une application en simplifiant la modélisation des interconnexions (modèle présenté en haut de la figure 3.13). En fait il s'agit d'abstraire ces interconnexions, masquant ainsi les détails tels que les signaux des protocoles. Cependant, l'hétérogénéité potentielle des protocoles de communication du réseau d'une part et des composants interconnectés d'autre part, n'est pas adressée. En effet, les composants et le réseau de communication sont supposés compatibles et les seuls adaptateurs matériels générés sont très minimalistes. Ainsi des inverseurs peuvent être mis en place pour adapter la polarité de signaux (en se basant sur le nom des signaux).
Coral peut donc aisément traduire cette description abstraite de l'architecture, utilisant des vues « virtuelles » des composants (V_i) et connexions, en une description physique grâce à un remplacement de ces objets virtuels par une représentation de leur ressource respective d'implémentation (R_i dans l'architecture en bas de la figure 3.13). Les adaptations éventuelles des interconnexions sont réalisées par de petits bloc de logique combinatoire (GL).
Cette méthode n'est utilisée qu'avec des plate-formes à base du protocole *CoreConnect* d'*IBM*, son application à toute autre plate-forme n'a pas été démontrée.

Sonics Silicon Backplane µNetwork

Sonics[86] apporte un environnement de développement « *FastForward Development* » constitué essentiellement de deux outils : *SOCCreator* et *CoreCreator*. *SOCCreator* permet la conception de systèmes monopuces par l'assemblage de composants compatibles autour d'un bus de communication unique : le *Silicon Backplane µNetwork*. Pour ce faire, il génère un assemblage de commutateurs, les « Agents » pour connecter les composants et composer le bus. Cependant, ces « Agents » ont une interface compatible avec le protocole Open Core Protocol (OCP)[70]. Aussi doit-on concevoir le composant ou l'adapter pour le rendre compatible au protocole *OpenCoreProtocol*.

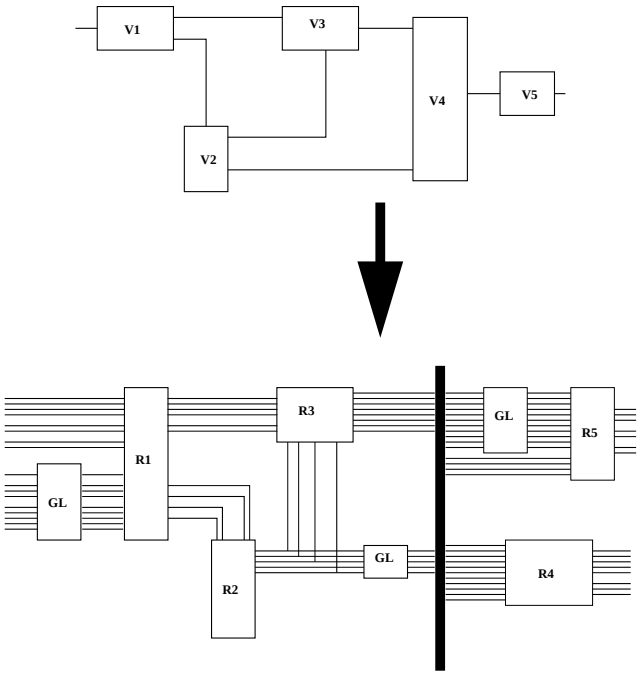


FIG. 3.13 – Raffinement d'interconnexions par IBM Coral.

SOCCreator accepte en entrée une description structurelle de l'architecture abstraite. Les instances de composants adaptées sont explicites, et sont interconnectées par des liens logiques. *SOCCreator* implémente ces liens par un bus micro-commuté le *Silicon Backplane μNetwork* (cf. 2.3.2 et figure 2.4).

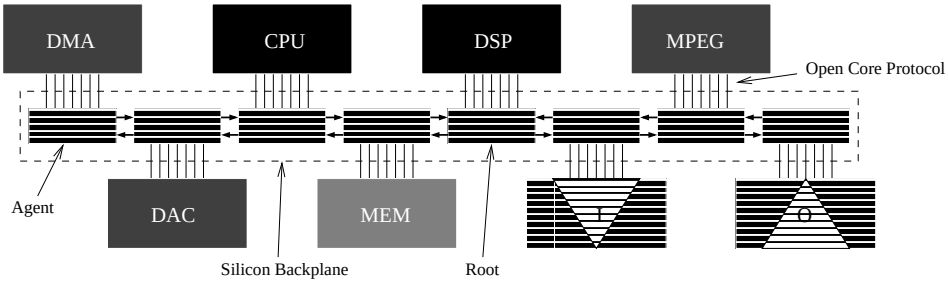


FIG. 3.14 – Architecture typique à base de Sonics μNetwork.

La figure 3.14 illustre une architecture typique organisée autour d'un réseau Silicon Backplane. Les différents items de cette architecture sont :
– CPU : processeur de logiciel générique + ROM et RAM locales ;
– DSP : processeur de logiciel de traitement de signal + ROM et RAM locales ;
– DMA : contrôleur d'accès directs aux mémoires, accélérant les transferts de données ;
– MPEG : processeur matériel permettant la décompression de vidéo ;
– DAC : processeur matériel permettant la conversion de données numériques en un signal analogique ;
– MEM : bloc de mémoire partagée, à accès uniformes ;
– I : ports d'entrées permettant la lecture d'informations provenant de capteurs externes ;
– O : ports de sorties permettant la commande d'actionneurs externes ;

Les bus systèmes

Certains fournisseurs de composants réutilisables offrent des modèles de bus systèmes compatibles avec un grand nombre de leurs composants. Ces bus s'élèvent de facto au rang de standards, spécifiques à une chaîne de développement, une famille de composants. Ainsi, les bus suivants sont copieusement utilisés pour interconnecter un

jeu de composants aux protocoles de communication compatibles. IDT a développé son bus [42] pour la connexion de périphériques, son utilisation n'est donc pas orientée multi-maîtres donc multi-processeurs ; Le *Peripheral Interconnect bus (PIbus)*[69] d'OMI, et *AMBA*[1] de ARM Inc. sont des bus partagés multi-maîtres ; Le standard *PCI* bien connu des architectes d'ordinateur personnel, est aujourd'hui décliné en version embarquée comme [59, 71].

Les standards d'interfaces

Afin de faciliter l'intégration de divers composants au sein d'une même architecture, plusieurs standards de protocoles et interfaces ont été érigés pour que lors de la conception de ces composants, ces protocoles soient adoptés. On trouve ainsi :

– WISHBONE[85, 82, 83, 84] d'OpenSilicore
– Le consortium *Virtual Socket Interface Alliance (VSIA)*[51, 81] qui définit un standard de documentation et de spécification *SLIF*[102, 103], un standard d'abstraction des interfaces de bus *O*n-*C*hip *Virtual Component Interface* et une représentation standardisée des types de données, le System-Level Data Types Standard.
– OpenCore[70] plébiscité par Sonics ;
– CoreConnect[41] d'IBM.

Ces standards offrent une manipulation et une utilisation simplifiées des composants grâce à leurs compatibilité directe. Cependant, ils ne répondent pas toujours pleinement aux besoins spécifiques des applications. Leur respect peut conduire à l'implémentation d'un jeu de fonctionnalités qui ne sont pas toutes utilisées. Leur utilisation reste faiblement répandue, car les utilisateurs potentiels leur reprochent les points précédemment cités ou alors la stratégie ou la politique des entreprises utilisatrices ne s'accommode pas des décisions des organismes de standardisation souvent « orientées » par d'autres entreprises. Des solutions propriétaires leur sont donc préférées.

Les travaux présentés par [38, 39] abordent la co-conception de système électronique sous la forme d'un flot de conception descendant. Le modèle d'entrée est un ensemble de tâches concurrentes communicant par des primitives de « Kahn » (lecture et écriture blocantes) au travers de files. Une action de partitionnement permet de spécifier une assignation des tâches aux composants logiciels ou matériels d'une plateforme prédéfinie comportant un unique bus système comme réseau de communication. L'implémentation de la solution retenue est réalisée par le ciblage manuel sur la plateforme aidé par la réutilisation de blocs d'adaptation de communication en bibliothèque.

Ils apportent aussi une structuration des interfaces logicielles $\leftrightarrow$ matérielles. Leur implémentation est abordée par la définition d'une architecture des adaptateurs matériels statiquement définis et permettant l'adaptation des composants compatibles avec le protocole VCI de VSIA au réseau de communication constitué de bus PIbus de OMI [69]. Malheureusement, ces travaux n'abordent pas l'automatisation de la synthèse de ces adaptateurs pour d'autres configurations matérielles (composant non compatible avec VCI, ou bus différent du PIbus).

3.4 Roses : l'intégration de composants autour d'un réseau de communication générique.

Roses est un flot de conception développé par le groupe *SLS* du laboratoire *TIMA*. Il permet l'intégration de composants matériels et logiciels au sein d'architectures. Celles-ci s'articulent autour de réseaux de communication appréhendés comme des composants génériques. Cette approche peut être définie comme l'assemblage de composants préconçus, afin que leur organisation apporte de façon spécialisée les ressources matérielles nécessaires à l'implémentation du système.

La figure 3.15 donne une vision simplifiée de l'intégration de composants autour de modèles génériques de réseaux de communication, telle qu'elle est réalisée par *Roses*. Les entrées de ce flot sont :

1. un modèle ou une spécification de l'application contenant une description macroscopique de la structure de l'architecture matérielle/logicielle d'implémentation : la *Macro-architecture* de l'application ;
2. une bibliothèque ouverte de modèles d'implémentation configurables pour 1^o les composants de calcul, 2^o les mémoires et 3^o les réseaux de communication ;
3. une bibliothèque d'implémentation de primitives d'adaptation permettant par composition de synthétiser les adaptateurs de communication.

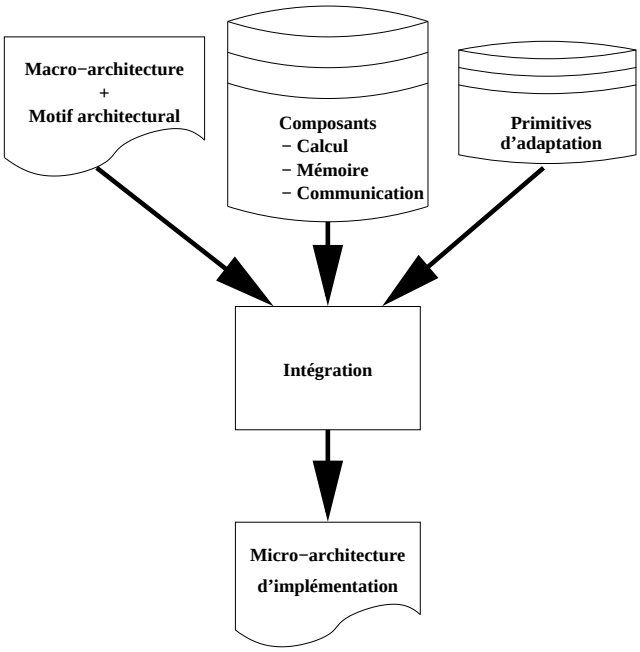


FIG. 3.15 – Intégration avec réseau de communication générique par *Roses*.

Il résulte de l'application de cette étape d'intégration sur les entrées suscitées un modèle *Micro-architectural* de l'application pouvant être pris en charge par un flot classique et commercial de synthèse matérielle RTL.
Dans la suite de cette section, le modèle interne de *Roses* pour la représentation d'architecture sera présenté en 3.4.1, la sous-section 3.4.3 présentera le flot complet de *Roses* alors que l'accent sur les spécificités de *Roses* qui le distinguent des autres outils d'intégration sera porté par la sous-section 3.4.4.

3.4.1 Le langage COLIF

CODESIGN LANGUAGE INDEPENDANT FORMAT (COLIF) est un langage de spécification développé au sein de l'équipe SLS[18]. Il permet la description topologique d'une application ainsi que la modélisation hétérogène en langages et en niveaux d'abstraction du comportement de ses fonctionnalités.

Concepts de base

COLIF offre trois concepts de base nécessaires à toute représentation topologique.
Ce sont :
– le *module* qui représente une unité fonctionnelle de calcul pouvant être hiérarchique.
– le port pour modéliser la connexion entre l'intérieur d'un module (sous-structure ou comportement) et son environnement externe. Il peut être hiérarchique afin d'abstraire des détails d'implémentation ou de spécification.
– le *net* représentant une unité de routage et transport de communication entre plusieurs *port*s de un ou plusieurs *module*s. Les détails d'implémentation tels que les signaux spécifiques au protocole de communication peuvent être cachés par une définition hiérarchique des *net*s offrant ainsi une abstraction des canaux de communication.

Ces trois concepts sont déclinés en trois classes :
– les définitions de modules, nets et ports sont respectivement abordés par l'utilisation de **MODULE**s, **NET**s et **PORT**s.
– Pour spécifier la structure interne de modules, ports et nets, il est possible de déclarer l'utilisation de sous-modules, sous-ports et sous-nets grâce aux **MODULE_DECL**s, **PORT_DECL**s et **NET_DECL**s.
– L'utilisation des définitions est offerte par les **MODULE_INSTANCE**s, **NET_INSTANCE**s et **PORT_INSTANCE**s qui sont des copies de **MODULE**s, **NET**s et **PORT**s spécialisées par :
  – L'adjonction d'un nom unique dans le niveau hiérarchique courant ;
  – la spécialisation hiérarchique des champs génériquement définis par les déclarations de sous-modules, sous-ports et sous-nets.

Enveloppe générique pour l'exécution et la réalisation d'interfaces pour architectures multiprocesseurs

Les concepts de *Colif* permettent la modélisation structurelle d'architecture. Une sémantique novatrice leur sont associée afin de modéliser des hétérogénéités diverses. Ces hétérogénéités seront introduites, la sémantique des « enveloppes » sera présentée.

Hétérogénéité des modèles : Les composantes d'un système peuvent être décrites de manières si diverses que leur composition se révèle périlleuse. Lors de la conception d'un système complet, il n'est pas toujours possible de faire progresser le traitement de toutes ses composantes de façon uniforme et parallèle. Ce fait peut être aggravé par :
– La réutilisation de composants dont on ne dispose pas des modèles au niveau d'abstraction courant ;
– Une méthodologie de conception modulaire, dans laquelle les concepteurs se concentrent sur les modules de façon individuelle, ce qui leur permet d'explorer un espace de solution conséquent sans être noyés par des spécifications trop volumineuses.

Pour des raisons évidentes de productivité, les méthodologies modernes de conception font massivement appel à la réutilisation de composants existants. Mais ces composants ayant été définis et conçus séparément, leur interconnexion n'est pas une opération triviale car rien ne garantit l'équivalence des protocoles de communication utilisés de part et d'autre. Il est malheureusement bien souvent le cas où de tels composants ne pourront être associés en raison de cette incompatibilité de protocoles.
Chaque composante d'un système peu appartenir à un domaine d'application spécifique (contrôle, traitement du signal, analogique). Or le concepteur de cette composante peut exiger l'utilisation locale d'un langage et d'un environnement de conception dédiés. Ainsi, un système peut être globalement défini par un jeu de sous-systèmes spécifiés au travers de langages spécifiques.
Bien que ces trois problèmes soient déjà suffisamment difficiles, les concepteurs spécifient leurs systèmes en les combinant. Ils utilisent des composants et des canaux de communication définis par des langages différents, à des niveaux d'abstraction différents et communicant par des protocoles incompatibles.
Bien loin d'être motivée par une recherche de la complexité, cette approche est justifiée par la nécessité de disposer à tout moment d'une spécification cohérente du système, autorisant une exploration d'architecture et un développement concurrent.

Enveloppes Par la possibilité de définir des *module*s, des *net*s et des *port*s hiérarchiques, COLIF permet, si on lui adjoint une sémantique spécifique, de spécifier des systèmes hétérogènes en abstraction et en communication. Cette sémantique additionnelle consiste en la définition de trois concepts : le *module virtuel*, le *port virtuel* et le *canal virtuel*.

– Le *module virtuel* permet d'adapter virtuellement un *module* à son environnement. Pour ce faire, on définit un *module* COLIF dans lequel on instancie le *module* à adapter. L'adaptation est alors prise en charge par les ports du *module virtuel* : les *port*s *virtuel*s.
– Le *port virtuel* a une hiérarchie particulière. Il est composé de :
  – un ensemble de ports *interne*s uniquement accessibles par le contenu (comportement ou structure) du module à adapter. Il s'agit en fait des ports natifs de ce module.
  – un jeu de ports *externe*s compatibles en abstraction et protocole avec les canaux de communication connectés.

  La granularité de ses sous-ports a la sémantique suivante : chaque sous-port participe et suffit à une même communication. Un ensemble de ports *RTL* implémentant le protocole d'une même communication, sont regroupés au sein d'un unique sous-port.
– Le *canal virtuel* permet l'abstraction des communications par l'encapsulation de nets, mais aussi de regrouper l'implémentation de plusieurs canaux sur un même jeu de ressources.

Exemple de spécification hétérogène d'un système Ainsi, grâce à COLIF une spécification de système telle que l'illustre la figure 3.16 peut être exprimée. Ce système comporte quatre instances de modules : **M1**, **M2**, **M3** et **M4**. Ce système est bien hétérogène en niveaux d'abstraction car :
– **M1** est décrit au niveau *RTL*, par le biais d'une machine d'états finie synchrone ;
– **M2** est structurellement défini par deux tâches logicielles concurrentes communicant entre elles et avec leur environnement par des messages (niveau *Message*) ;

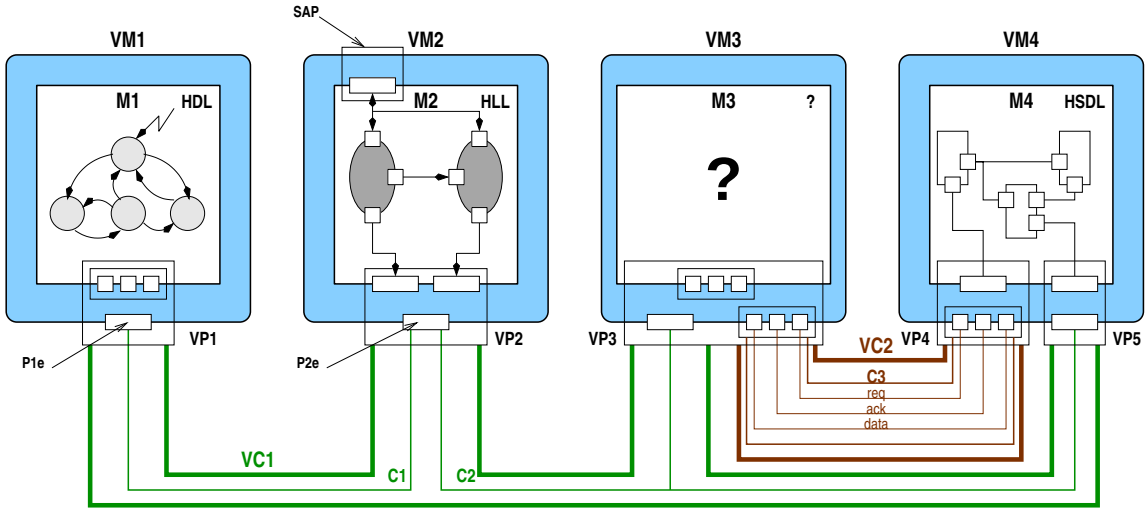


FIG. 3.16 – Spécification hétérogène de système.

– Seule l'interface de **M3** est connue, il s'agit là d'un composant protégé (propriété intellectuelle) qui sera appréhendé comme une boîte noire. Étant donné la granularité utilisée pour spécifier cette interface, le comportement de **M3** est supposé décrit au niveau d'abstraction *RTL* ou *Comportemental détaillé*.
– **M4** contient une hiérarchie de sous-modules n'ayant pas encore de connotation architecturale (matériellement parlant). Cette structure est décrite au niveau *Transaction*.
– Les canaux de communication **C1** et **C2** sont décrits au niveau *Message* alors que **C3** est décrit au niveau *RTL*.

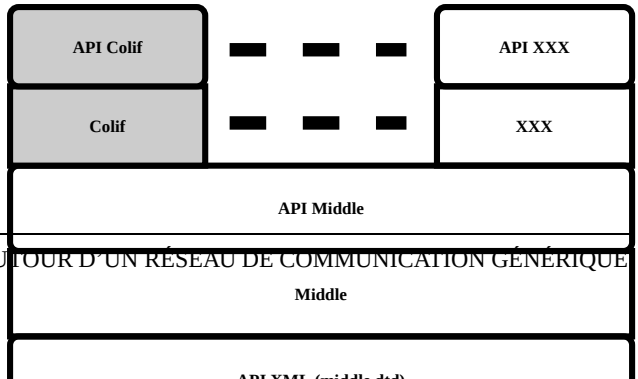
Cette spécification est aussi multi-langage :

– Un langage de description matérielle (*HDL*) tel que VHDL ou VERILOG est utilisé pour spécifier **M1** ;
– Un langage haut niveau de description logicielle (*HLL*) tel que C++ ou JAVA est utilisé pour spécifier **M2** ;
– Le langage de description de **M3** est laissé à la discrétion de son concepteur. En général, pour de tels composants, seuls des modèles de simulation destinés à être remplacés par des modèles de niveau *Circuit* lors de l'étape de Placement-Routage sont disponibles.
– **M4** est modélisé par un algorithme exprimé par un langage de co-conception (*HSDL*) tel que SYSTEMC.

Pour garantir la cohésion de modèles aussi hétérogènes, des enveloppes virtuelles d'adaptation sont créées : les **Modules Virtuels** et les **Canaux Virtuels**. Alors que les premiers permettent l'encapsulation de n'importe quel langage et niveau d'abstraction, les seconds ne sont que des artifices permettant de regrouper hiérarchiquement tous les canaux potentiellement candidats à une implémentation utilisant un même jeu de ressources (e.g. un bus partagé). Ainsi sur la figure 3.16, **M1** est instancié en interne au module virtuel **VM1** adaptant virtuellement les ports *RTL* de **M1** au canal de communication *Message* **C1** par le port virtuel **VP1**. Cette adaptation est sémantiquement réalisée par la hiérarchie de **VP1** qui fait correspondre à l'ensemble des ports *RTL* de **M1**, un unique port *Message* vu par l'extérieur : **P1e**.

Ces correspondances ne sont pas forcément bijectives. En effet plusieurs ports internes (appartenant à l'interface du module à encapsuler) peuvent être associés à un unique port externe. Le port virtuel **VP2** offre un exemple d'une telle association. Ce mécanisme, fort utile pour l'économie de connexions à un même média de communication, nécessite cependant une résolution du canal logique (**C1** ou **C2**) pour les communications sortantes, ainsi qu'une résolution du port interne cible (**P21** ou **P22**). Ces résolutions peuvent être opérées par l'ajout d'identifiant de canal et/ou de port sous la forme d'adresse virtuelle dans le paquet de communication. Le port virtuel **VP3** illustre une situation symétrique à celle résolue par le port **VP2**, avec comme difficulté supplémentaire l'utilisation de ports externes hétérogènes en abstraction.

Enfin des ports ne participant pas aux communications inter-modules peuvent être spécifiés. Désignés par le terme Service Access Port (SAP), ils permettent de spécifier des ressources matérielles destinées à être implémentées par des périphériques de processeur (coprocesseur, accélérateur, temporisateur, etc.) ainsi que leur partage par les différentes tâches



logicielles. Le port **SAP** de la figure 3.16 permet de spécifier l'adjonction d'un temporisateur à **VM2** afin de pouvoir mesurer des intervalles de temps avec précision.

3.4.2 Construction du langage COLIF

Le langage COLIF est utilisé dans ces travaux pour représenter des topologies architecturales. Cette sémantique structurelle peut localement être enrichie par l'interprétation de paramètres. Une sémantique propre aux bibliothèques de l'outil de raffinement lui est adjoint. Les concepts propres à COLIF ont déjà été introduits en 3.4.1. Sa construction a été abordée par couches. Ces couches sont :

– une couche d'interfaçage du langage COLIF avec le code applicatif (outil de CAO) : `API Colif`. Elle fournit un jeu de primitives permettant de découpler l'utilisation par un outil des structures de données sous-jacentes, de l'implémentation de ces structures et de leur gestion ;
– une couche d'implémentation du langage COLIF. Elle définit les structures de données permettant la représentation d'architecture ;
– un langage et son API permettant la définition de type de structure de données et l'utilisation de ces structures : `Middle` et `API Middle` ;
– le langage XML et son interface.

Le langage « Extensible Mark-up Language (XML) » est un langage à balises extensibles, permettant la structuration de données. Une restriction de l'utilisation de XML est opérée par un jeu de balises formant un modèle générique de document ou « Data Type Document (DTD) ». Cette restriction, dénommée « Mark-up internal data-description language extension (Middle) », offre le pouvoir d'expression manquant à XML pour exprimer en quelques dizaines de balises les différents concepts de COLIF. La contribution de ces travaux relative à l'élaboration de ces langages de représentation est localisée sur la spécification de COLIF[19]. La conception de Middle est une contribution de [29]. Le langage COLIF permet la modélisation d'architecture par l'offre de trois concepts de base : 1^o le module, 2^o le port et 3^o le net. Les descriptions hiérarchiques sont traitées par la possibilité de déclarer l'utilisation de ces concepts de façon récursive. Ainsi un module, un port et un net peuvent respectivement contenir la déclaration de l'utilisation d'un sous-module, d'un sous-port et d'un sous-net. Ces déclarations n'ont pour objectif que la définition d'une structure hiérarchique et générique. Elles ne figent pas les dimensions des ports et des nets, ni la valeur des paramètres à sémantique contextuelle. Les relations existantes entre les concepts de bases sont illustrées par la figure 3.18 qui en présente la définition et l'utilisation des déclarations sous la forme d'un graphe de classes UML. Une fois les modules et ports et nets hiérarchiquement définis, il est possible de les instancier, de les connecter et de les spécialiser grâce aux instances de ces trois concepts illustrées en figure 3.19. La sémantique de ces classes est brièvement introduite par le tableau ci-dessous.

TAB. 3.2 – Sémantique des classes COLIF.

`Classe`	
Responsabilité	
`Membre`	Signification
`MODULE`	
Modélise un module.	
`description`	Brève description du module.
`level`	Niveau d'abstraction du module.
`entity`	Référence vers une description de l'interface du module.
`content`	Référence vers une description du contenu du module.
`MODULE_ENTITY`	
Modélise l'interface d'un module et en qualifie la nature.	

suite sur la page suivante.

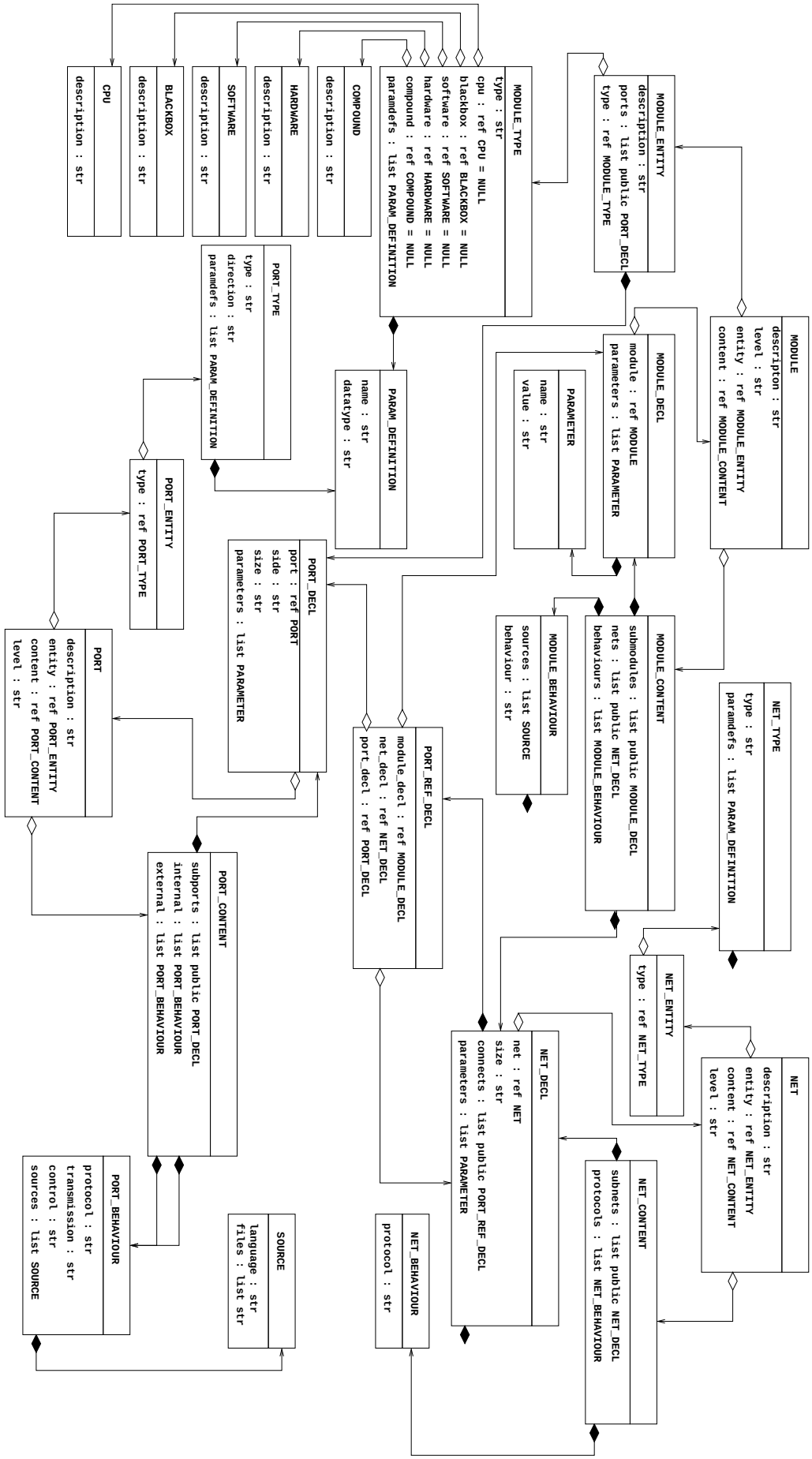


FIG. 3.18 – Définition et déclarations des concepts COLIF.

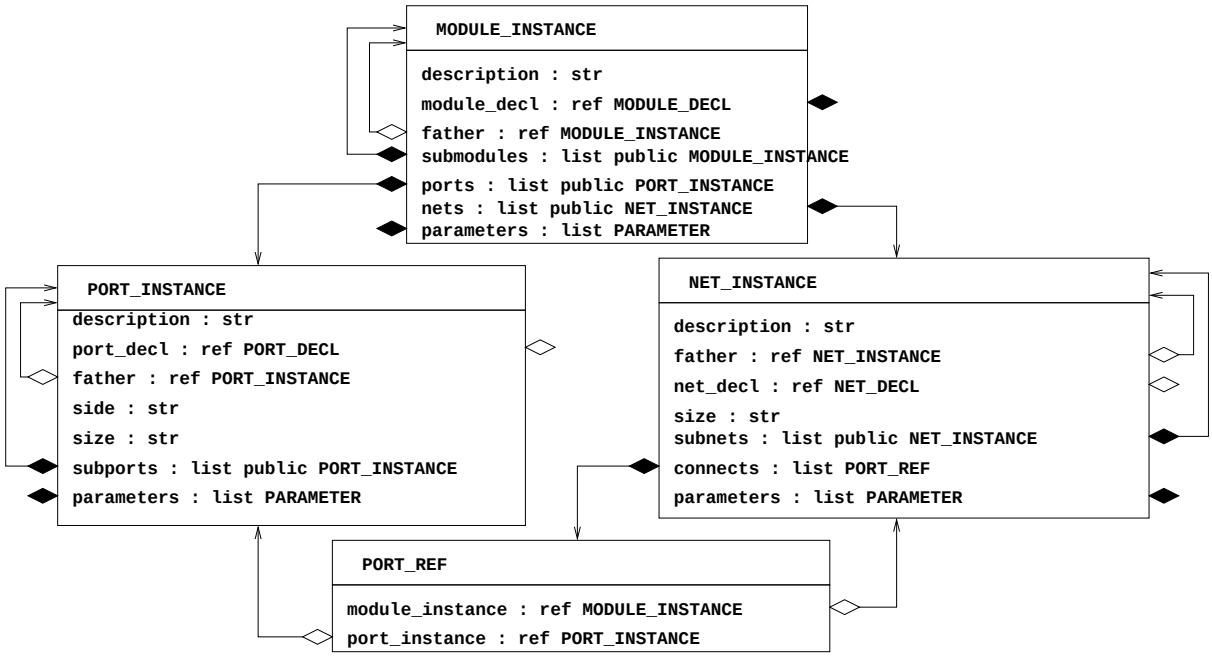


FIG. 3.19 – Instances des concepts COLIF.

suite de la page précédente.	
`description`	Brève description de l'entité du module.
`ports`	list de déclaration de ports définissant l'interface du module.
`type`	Identification de la nature du module (sémantique superposée permettant de une famille de CPU ou IP HW).

`MODULE_DECL`	
Déclaration de l'utilisation d'un module au sein d'un module de niveau hiérarchique supérieur.	
`module`	Référence vers la définition du module utilisé.
`parameters`	Liste de paramètres permettant d'enrichir la description du module de caractéristiques non-fonctionnelles et non-structurelles (valeurs par défaut).

`MODULE_CONTENT`	
Modélise le contenu d'un module.	
`submodules`	Liste des déclarations de modules internes.
`nets`	Liste des déclarations des nets internes.
`behaviours`	Liste des descriptions de comportements adjoints au module.

`MODULE_TYPE`	
Caractérise la nature d'un module.	
`type`	Nature du module, parmi `cpu`, `blackbox`, `software`, `hardware` et `compound`.
`cpu`	Référence vers un identifiant de processeur.
`blackbox`	Référence vers un identifiant d'IP matériel.
`software`	Référence vers un identifiant de modèle de calcul logiciel.
`hardware`	Référence vers un identifiant de modèle de calcul matériel.
`compound`	Référence vers un identifiant de description structurelle.
`paramdefs`	Liste de définitions de `parameters`.

`MODULE_BEHAVIOUR`
Caractérise le comportement d'un module feuille de la hiérarchie.
suite sur la page suivante.

suite de la page précédente.	
`sources`	Liste de groupes de descripteurs de fichiers sources contenant une modélisation du comportement du module.
`behaviour`	Identifiant du comportement.

`NET`	
Modélise un net d'interconnexion.	
`description`	Brève description du net.
`entity`	Référence vers une caractérisation de la nature du net.
`content`	Référence vers la description du contenu du net.
`level`	Niveau d'abstraction du net.

`NET_ENTITY`	
Qualifie la nature d'un net.	
`type`	Référence vers un identifiant de la nature du net.

`NET_CONTENT`	
Modélise le contenu d'un net.	
`subnets`	Liste des déclarations de nets internes.
`protocols`	Liste des protocoles supportés par le net.

`NET_TYPE`	
Caractérise la nature des informations transitant sur un net.	
`type`	Identifiant des informations véhiculées par le net, tel que `control`, `data` et `event`.

`NET_DECL`	
Déclaration de l'utilisation d'un net au sein d'un net de niveau hiérarchique supérieur (tel qu'un canal abstrait).	
`net`	Référence vers la définition du net.
`size`	Taille en bits des informations véhiculées (valeur par défaut).
`connects`	Liste des ports connectés à ce net.
`parameters`	Liste de paramètres permettant d'enrichir la description du net de caractéristiques non-fonctionnelles et non-structurelles (valeurs par défaut).

`NET_BEHAVIOUR`	
Caractérise le comportement d'un net.	
`protocol`	Identifiant du protocole du net.

`PORT`	
Modélise un port.	
`description`	Brève description du port.
`entity`	Référence vers une qualification du port.
`content`	Référence vers une description interne du port.
`level`	Niveau d'abstraction du port.

`PORT_ENTITY`	
Qualifie la nature d'un port.	
`type`	Référence vers un identifiant de la nature du port.

`PORT_CONTENT`	
Modélise le contenu d'un port.	
`subports`	Liste des déclarations de ports internes.
`internal`	Liste des accès internes au module.
`external`	Liste des accès externes au module.

`PORT_TYPE`
Caractérise la nature des données transitant au travers d'un port.

suite sur la page suivante.

suite de la page précédente.	
`type`	Identifiant des informations transitant par le port, tel que `control`, `data` et `event`.
`direction`	Direction du port, parmi `in`, `out`, `inout` et `void`
`paramdefs`	Liste des définitions de paramètres pouvant annoter le port de caractéristiques non-fonctionnelles et non-structurelles.

`PORT_BEHAVIOUR`	
Caractérise le comportement d'un port.	
`protocol`	Identifiant du protocole supporté par le port.
`transmission`	Caractérisation du partage du net, parmi `point-to-point`, `multicast`.
`control`	Caractérisation de la synchronisation des accès au port, parmi `synch` et `asynch`
`sources`	Liste des groupes de fichiers sources modélisant le comportement du port.

`PORT_DECL`	
Déclaration de l'utilisation d'un port au sein d'un port de niveau hiérarchique supérieur (tel qu'un port abstrait).	
`port`	Référence vers la définition du port.
`side`	Accessibilité au port pour le contenu du module, parmi `internal`, `external` et `both` (valeur par défaut).
`size`	Taille en bits des informations transitant par le port (valeur par défaut).
`parameters`	Liste de paramètres permettant d'enrichir la description du port de caractéristiques non-fonctionnelles et non-structurelles (valeurs par défaut).

`MODULE_INSTANCE`	
Instance de module.	
`description`	Commentaires.
`module_decl`	Référence à la déclaration de module spécialisée par cette instance.
`father`	Référence à l'instance de module de niveau hiérarchique supérieur et contenant l'instance courante.
`submodules`	Liste des sous-instances de modules internes.
`ports`	Liste des instances de ports composant l'interface de cette instance de module.
`nets`	Liste des des sous-instances de nets internes.
`parameters`	Liste de `PARAMETER`s assignés de valeurs spécifiques à l'instance.

`PORT_INSTANCE`	
Instance de port.	
`description`	Commentaires.
`port_decl`	Référence à la déclaration de port spécialisée par cette instance.
`father`	Référence à l'instance de port de niveau hiérarchique supérieur et contenant l'instance courante.
`side`	Accessibilité du port (valeur spécifique à l'instance).
`size`	Taille en bits des informations transitant par le port (valeur spécifique).
`subports`	Liste des instances de ports internes.
`parameters`	Liste de `PARAMETER`s assignés de valeurs spécifiques à l'instance.

`NET_INSTANCE`	
Instance de net.	
`description`	Commentaires.
`father`	Référence à l'instance de net de niveau hiérarchique supérieur et contenant l'instance courante.
`net_decl`	Référence à la déclaration de net spécialisée par cette instance.
`size`	Taille en bits des informations véhiculées par le net (valeur spécifique).
`subnets`	Liste de sous-instances de net internes.

suite sur la page suivante.

suite de la page précédente.	
connects	Liste des instances de ports connectées à cette instance de net.
parameters	Liste de PARAMETERs assignés de valeurs spécifiques à l'instance.

PORT_REF	
Référence à une instance de port.	
module_instance	Référence à l'instance de module connectée.
port_instance	Référence à l'instance de port connectée.

Une description plus détaillée de ces modèles est offerte par [24].

3.4.3 Architecture du fbt

La figure 3.20 présente le flot de conception mis en place par l'équipe System Level Synthesis (SLS). Il part d'une spécification architecturale et comportementale du système à concevoir. La description est alors raffinée de niveau d'abstraction en niveau d'abstraction. En sortie, nous obtenons le code logiciel et matériel réalisant l'application.

Ce flot débute au niveau *message*, donc après que le partitionnement logiciel/matériel ait été décidé. Il se termine au niveau *micro-architecture*, où une classique étape de compilation et de synthèse logique permet d'obtenir la réalisation finale du système. C'est un flot descendant qui permet cependant une validation à tous les niveaux et de revenir en arrière à chaque étape.

En entrée, le flot attend une description de l'application au niveau *message* (la *Macro-architecture*) :

– Le comportement est modélisé par un ensemble de tâches concurrentes dont les algorithmes sont exprimés en divers langages hétérogènes de modélisation[64].
– La communication est modélisée par des envois de transactions entre composants de calcul ou entre composants de calcul et mémoires partagées.
– La structure peut, suivant le niveau d'abstraction, donner un modèle macroscopique de l'application, où les fonctionnalités implémentées par une même ressource sont hiérarchiquement regroupées. Ces groupes ou blocs hiérarchiques sont annotés de paramètres de raffinement guidant les outils dans le choix et la spécialisation des composants à mettre en œuvre. On trouve ainsi des modules et des canaux de communication hiérarchiques qui regroupent des instances fonctionnelles ou des communications concurrentes destinées à être implémentée par une unique ressource matérielle.

Cette description est traduite dans la forme intermédiaire COLIF pour être raffinée au cours de trois étapes :

– La première étape fait passer la description au niveau *message* (cf. 3.2.3) : elle effectue la synthèse de la communication et les allocations globales de la mémoire.
– La deuxième étape génère une table d'allocation : elle effectue les affectations de mémoire et de protocoles, et également les optimisations des accès.
– La dernière étape fait passer la description au niveau micro-architecture (cf. 3.2.3) : elle effectue la génération des interfaces logicielles et matérielles qui permettent l'assemblage des divers composants ainsi que leurs communications.

Les paragraphes suivants vont présenter les différentes étapes de ce flot :

– La spécification d'un système électronique par un modèle de niveau transaction (cf. 3.2.3) ;
– L'allocation de ressources assignées à l'exécution de groupes de sous-fonctionnalités de l'application : le partitionnement. Il s'agit de regrouper les tâches, d'allouer et d'assigner des ressources de calcul, de mémorisation et de communication et enfin d'ordonnancer l'usage de ces ressources par toutes les fonctionnalités les partageant.
– La validation des décisions architecturales par l'exécution du modèle de niveau *message* ;
– La génération pour chaque CPU alloué d'un système d'exploitation spécifique[29] implémentant tous les services requis par les tâches logicielles se le partageant ;
– La génération d'une Architecture Locale (AL) pour chaque processeur (logiciel ou matériel) alloué contenant les unités d'adaptation de communication requises ;
– La génération de modèles concrets des réseaux de communication ;
– La validation de la *micro-architecture* par Co-Simulation.

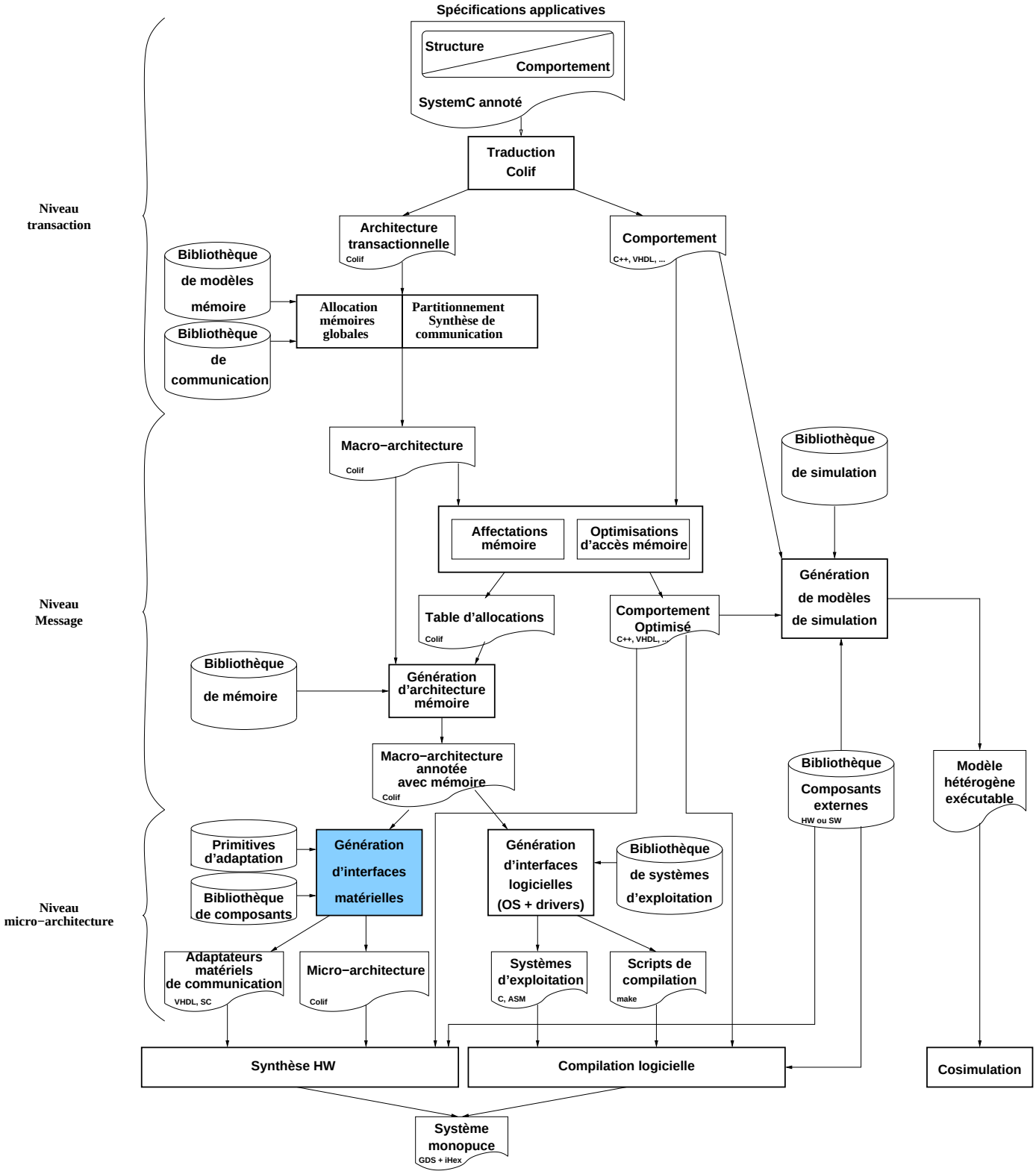


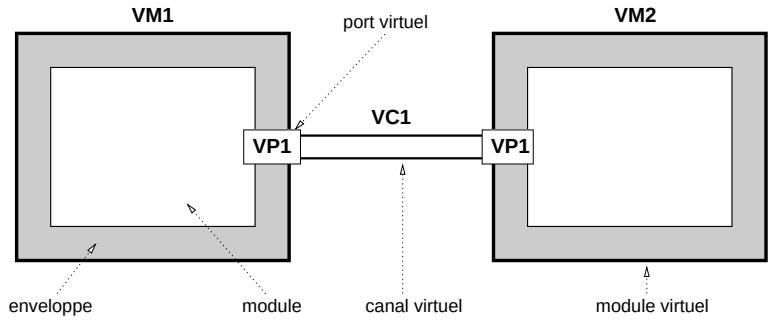
FIG. 3.20 – Flot de conception de l'équipe SLS.

L'entrée du flot au niveau transactionnel

En entrée, *Roses* accepte une description dans le langage SYSTEMC[90, 91] enrichi des concepts de *Module Virtuel*, de *Port Virtuel* et de *Canal Virtuel* (cf. 3.4.1). Le langage *Virtual Architecture Description Language (VADeL)* ainsi obtenu, permet de décrire la structure et le comportement d'un système hétérogène. *VADeL* apporte à SYSTEMC le

```
// File main.cc
#include <iostream.h>
#include <stdlib.h>
#include <vadel.h>
#include "appli_types.h"
#include "vm1.h"
#include "vm2.h"
/------------------------------------
// definition des canaux virtuels
VA_CHANNEL(vc1)
{
   va_ch_mac_pipe<long int>   CH2;
  VA_CCTOR(vc2) {VA_CEND};
};
/------------------------------------
// Programme principal
int sc_main(int argc, char *argv[])
{
   // Initialisation
   va_init();
   // instancie les canaux virtuels
   vc1 VC1("VC1");
   // instancie les modules virtuels
   vm1 VM1("VM1");
   vm2 VM2("VM2");
   // connecte les ports
   (*VM1.VP1)(VC1);
   (*VM2.VP1)(VC1);
   // Traduction vers COLIF
   sc_start(0);
   return EXIT_SUCCESS;
}
```

(a) Code VADeL



(b) Structure

FIG. 3.21 – Exemple de spécification d'entrée VADeL.

support multilangage et formalise les hétérogénéités en niveaux d'abstraction et en protocoles grâce aux enveloppes. Cette description peut être effectuée au niveau *Transactionnel*, ou aux niveaux inférieurs (cf. 3.2). Il est aussi possible de combiner les niveaux. Elle donne la structure, et éventuellement (ce n'est pas le cas des IP) le comportement de composantes mixtes[2]. Si certains composants sont fournis sans comportement, ils sont considérés comme des IP ou boîtes noires.

La figure 3.21 donne un exemple d'une telle description : les modules **VM1** et **VM2** (instanciés en lignes 26 et 27, d'après les définitions importées en lignes 6 et 7) communiquent au travers de ports abstraits (*VP1*) interconnectés par un canal abstrait (instancié en ligne 24 d'après une définition en ligne 11). La réalisation de ces communications et l'allocation de ressources n'est pas encore définie et ce modèle-ci n'est donc pas encore exécutable.

Étape de traduction vers COLIF

Les évolutions des langages de modélisation aussi récents que SYSTEMC et les habitudes des utilisateurs rendent préférable l'utilisation de modèle décrit en un langage interne qu'un traducteur permet de découpler des langages d'entrée. C'est pourquoi la première étape du flot consiste à convertir la description *VADeL* en une description COLIF. Cette étape extrait les informations structurelles du modèle d'entrée, pour les mettre sous un format COLIF qui sera traité tout au long du flot. Les descriptions comportementales sont conservées séparément dans des fichiers référencés dans la description structurelle.

[2]Les composantes d'un système peuvent être de nature électronique numérique ou analogique, ou optique, ou mécanique [62]

Étape d'allocation mémoire, de partitionnement et de synthèse de la communication

Cette étape permet de passer d'une description du niveau *Transactionnel* au niveau *Message*. Elle se compose de quatre opérations :

1. L'allocation de ressources de calcul parmi des processeurs logiciels ou des accélérateurs matériels ;
2. L'assignation des fonctionnalités du modèle transactionnel aux ressources allouées ;
3. L'allocation mémoire ;
4. L'assignation des variables partagées aux blocs mémoires ;
5. La synthèse des communications impliquant l'allocation de média de communication et leur assignation à l'implémentation des canaux de communication transactionnels ;
6. La spécialisation et l'optimisation des accès mémoires.

Les trois paragraphes suivants présentent succinctement la contribution du groupe SLS pour ce domaine.

Partitionnement, allocation et assignation mémoire Les fonctionnalités de l'application sont regroupées selon des critères d'affinités tels que l'appartenance à un même domaine (ie. traitement du signal, etc. ...) et l'utilisation partagée de même données. Les groupes, ainsi constitués, se voient attribués une ressource d'exécution conciliant les contraintes de coûts et performances. Ces ressources sont choisies parmi les processeurs de logiciel ou blocs matériels préalablement défini (IP) appartenant à un portfolio de composants réutilisables. Cette étape reste très empirique du fait de la combinaison de ces objectifs : coûts réduits, performances suffisantes, etc. ... Au sein du flot *Roses*, cette étape est manuelle, mais bénéficie d'une méthode d'analyse statique [5] guidant le concepteur jusqu'à une solution acceptable. L'allocation mémoire consiste en la définition des blocs mémoires qui vont contenir les données de l'application. Cette étape basée sur la résolution d'un programme linéaire est présentée dans [54]. Dans ce travail, une heuristique a été définie afin de déterminer automatiquement une configuration optimale pour réduire le coût en communication mémoire. Une fois que les blocs mémoire ont été décidés, il est possible de leur assigner les données partagées.

Synthèse de la communication La synthèse consiste à choisir les protocoles de communication et les éléments de calcul (processeurs, ASIC, etc.) qui seront utilisés (cf. [63, 37]). Dans l'état actuel du flot, aucun outil ne permet d'effectuer cette synthèse de communication. Cette opération est donc effectuée à la main.
Des résultats de simulation au niveau *message* permettent de guider les choix pour les protocoles. À l'avenir, il est prévu d'intégrer une méthodologie et des outils permettant d'automatiser les choix à l'aide d'une bibliothèque de résultats de simulation [5].

Étape de spécialisation et d'optimisation des accès aux mémoires Cette étape, présentée dans [55], fait partie des travaux de thèse de Samy Meftali et de Ferid Gharsali. Elle consiste en :

1. le raffinement des méthodes d'accès aux variables partagées sous les contraintes architecturales définies par l'étape d'allocation ;
2. la restructuration des algorithmes afin de limiter le nombre d'accès.

Étape de génération d'architecture mémoire

La génération d'architecture mémoire, présentée dans [31], fait partie des travaux de thèse de Ferid Gharsali : les types de mémoires, leurs contrôleurs et leurs interfaces sont générés à partir d'une bibliothèque de mémoire. Le principe de cette génération est celui d'un assemblage de blocs de la bibliothèque (cf. [30]). Cet assemblage est guidé par les choix d'allocation et de synthèse.

Étape de génération d'adaptateurs matériels

La génération d'adaptateurs matériels permet d'interconnecter les divers éléments de calcul : en effet ces éléments ne sont pas tous compatibles entre eux. Cette étape, fait partie des travaux de cette thèse et sera développée dans le chapitre 4.

Étape de génération de systèmes d'exploitation

Les parties logicielles ne peuvent pas être exécutées directement sur les processeurs : l'exécution concurrente de plusieurs tâches sur un même processeur et les communications entre le logiciel et le matériel doivent être gérées par une couche logicielle appelée système d'exploitation. La génération de systèmes d'exploitation est adressée par les travaux de Lovic Gauthier [29].

Les étapes de validation

La validation d'un système peut se faire de diverses manières. Dans le flot *Roses*, seules deux approches sont développées :

1. exécution d'un modèle d'implémentation simulée ;
2. exécution d'un modèle d'implémentation transposée.

La première solution met en œuvre des outils permettant de reproduire fidèlement le comportement de l'implémentation du système : des simulateurs de processeurs et des simulateurs de composants matériels. La seconde permet, grâce au remplacement des technologies d'implémentation par des solutions plus souples et propices à l'itération, de réaliser à moindre effort le système et d'observer in fine son comportement in situ.

Par simulation

La génération d'enveloppes de simulation Le flot permet d'effectuer des simulations du système avec toutes les combinaisons possibles des niveaux d'abstraction vus en 3.2. La technique de base consiste à encapsuler les divers composants à simuler dans des enveloppes qui adaptent le niveau de leurs communications à celui de la simulation globale. Ces enveloppes permettent aussi d'adapter les divers simulateurs entre eux, comme par exemple un simulateur VHDL et un ISS. Cette étape, décrite dans [63], fait partie des travaux de thèse de Gabriela Nicolescu.

La cosimulation Grâce aux enveloppes, il est possible d'effectuer des validations par cosimulation pour toutes les étapes du flot, ou même sur des composants situés à des étapes différentes. Plus le niveau d'abstraction est élevé, plus la simulation est rapide et plus le niveau est bas et plus la simulation est précise. Cette cosimulation est multi-niveau, mais elle peut être aussi multi-langage. La raison de telles cosimulations est que très souvent un langage est adapté pour la simulation de certaines parties d'un système, mais inadapté pour d'autres. Par exemple les simulateurs [92, 56] sont adéquats pour les circuits mais pas pour la mécanique qui est mieux modélisée par MATLAB. Pour effectuer une cosimulation, les divers simulateurs sont exécutés par différents processus concurrents. Un processus de contrôle gère l'ensemble des communications et synchronisations entre les simulateurs par le biais de mémoires partagées et des signaux. Le modèle temporel utilisé est le modèle synchrone : chaque simulateur exécute un pas de calcul, puis toutes les données sont échangées grâce au processus de routage. Une fois que les communications sont achevées, un autre pas de calcul peut être effectué. Le processus de contrôle est décrit en SYSTEMC. Cela permet de l'utiliser pour décrire des parties matérielles ou logicielles au niveau *message*, sans avoir à utiliser un simulateur externe[3].

Par émulation Lorsque les niveaux d'abstraction des modèles à vérifier deviennent trop détaillés pour autoriser une validation performante et donc suffisante sous les contraintes de temps de développement, il est fait appel à l'émulation. L'émulation est une pratique issue de la recherche de performances pour les modèles de validation. L'accélération ciblée est très similaire à celle recherchée par la co-conception :

– les simulateurs sont des instances logicielles capables grâce à leur extrême reconfigurabilité de mimer divers comportements. Malheureusement cette généricité se paie par des performances restreintes. En effet, les expérimentations présentées dans [64](¶5.3.4) exhibent une performance de $\frac{0,4*(2*40.10^6+2*10^7)}{20*3600} \simeq 560$ cycles horloges simulés par seconde[4] pour une application quadriprocesseur au niveau RTL. Il en résulte une durée de 20 heures de calcul pour simuler 0,4 secondes de vie réelle.
– des unités matérielles, les émulateurs, permettent d'accélérer localement la validation. Pour ce faire on utilise :
  – soit des implémentations sur des architectures reconfigurables (FPGA) des modèles à valider ;

[3] ce qui permet d'avoir un gain notable en vitesse de simulation
[4] sur UltraSparc II, 333Mhz et 256 Mo.

– soit des implémentations finalisées et dédiées pour le test des composants réutilisés (dont la validation unitaire n'est plus une préoccupation).

Au sein du flot *Roses*, Arif Sasongko élabore une méthodologie d'émulation à base de *testchips*[5] de processeurs de logiciel et de FPGA pour émuler les processeurs matériels et les réseaux de communication.

Utilisation des résultats

Les mesures issues des étapes d'évaluation pourront servir de base pour mettre en place une méthodologie d'exploration [5] qui permettra dans un premier temps de guider les choix du concepteur dans l'étape de synthèse de l'architecture, puis d'automatiser cette synthèse avec la recherche d'optimum. Pour ce faire, il faudra définir des modèles de performances et de coûts, et des heuristiques permettant de les composer modulairement en même temps que l'on compose les divers modules des systèmes à générer.

3.4.4 Spécificités du flot de Roses

Le flot de raffinement d'architecture présenté dans ce mémoire peut être caractérisé par les modèles de systèmes acceptés en entrées, les modèles de systèmes générés et quelques hypothèses relatives aux spécifications.

Modèles d'entrée

Les systèmes à raffiner sont modélisés comme des réseaux de nœuds de calcul et de mémorisation : la *Message*. Les interconnexions peuvent y être modélisées par des canaux abstraits de communication ou par des IP matériels de niveaux RTL. On peut alors annoter chacune des composantes de cette spécification par des attributs modélisant les décisions de raffinement. Ces annotations permettent de spécifier les ressources de calcul et de communication à mettre en œuvre au sein de la *Micro-Achitecture*.

Sorties

Il résulte de l'application de ce flot sur un modèle de système tel que précédemment présenté :

1. Un modèle structurel de la *Micro-Architecture* explicitant les instances de processeurs, de mémoires, de décodeurs d'adresse et de coprocesseur de communication ainsi que la structure interne de ces derniers. Ce modèle est exprimé en COLIF et peut aisément être traduit en VHDL et SYSTEMC par l'utilisation de traducteurs développés dans le groupe SLS.
2. Des fichiers décrivant en VHDL et/ou en SYSTEMC le comportement *RTL* des décodeurs d'adresse, des adaptateurs de modules et des adaptateurs de canaux, sont générés.

Particularités du flot

Lors de l'application de ce flot, trois hypothèses sont formulées.
Les données partagées par plusieurs processeurs échappent au mécanisme de cache mémoire par assignation des canaux sur une plage mémoire non cachée. En fait, le modèle de calcul le plus souvent usité étant le «Data-Push» (envoi de messages du producteur au consommateur), ce mécanisme se révèle d'un intérêt réduit pour les architectures spécifiques aux applications (canaux de communication définis et fixés). De plus, par rapport aux systèmes sur cartes, l'utilisation de caches est bien moins intéressante. En effet, les accès aux mémoires sont bien plus uniformes et performants grâce à :

1. l'absence des fortes capacités des pads d'entrées/sorties ;
2. l'utilisation de ressources de communication à bandes passantes bien plus élevées.

La reconfiguration d'architecture en cours d'exécution de l'application n'est pas notre objectif. Cette hypothèse impose donc comme spécification d'entrée un réseau de nœuds de calcul statiquement défini. Ceci se traduit en une allocation et une assignation statiques des ressources aux diverses tâches applicatives.
Le nombre de fonctionnalités destinées à une implémentation logicielle ne cessant de croître, les architectures tendent à contenir un plus grand nombre de processeurs et donc un plus grand nombre de canaux. Si le nombre de processeurs

[5] implémentation sur technologie ASIC d'un core de processeur et de logique additionnelle de test.

augmente c'est bien évidemment pour s'associer au parallélisme de l'application et pouvoir respecter des contraintes temporelles de plus en plus courtes. Mais l'augmentation du nombre de canaux de communication inhérente à la répartition géographique de plus en plus vaste des nœuds de calcul fait naître le paradoxe suivant : les processeurs passent plus de temps à s'échanger des données qu'à réellement calculer les algorithmes applicatifs.
Aussi le parti d'utiliser un coprocesseur de communication de complexité variable afin de pouvoir régler la répartition de charge d'un processeur entre son calcul applicatif et ses communications, est adopté.

Contributions du flot Roses à la conception de systèmes monopuces

Ce flot offre une automatisation de la conception depuis l'étape de spécification *Message* jusqu'à la spécification de niveau *Micro-Architecture*.
Cette contribution se décompose en trois actions. La première étant la génération d'un modèle de niveau *micro-Architecture*, spécifique à l'application, constitué de plusieurs instances de processeurs et composants périphériques nécessaires à leur mise en œuvre (mémoires, décodeurs d'adresses, temporisateur, coprocesseur de communication ...). La seconde de ces actions est la synthèse d'adaptateurs de communication spécifiques aux communications de chaque nœud de calcul : les coprocesseurs de communication. Et enfin, le raffinement du réseau de communication intervient en troisième et dernière action.
Les différentes étapes chronologiques de ce flot de conception sont :

1. la spécification initiale du système pouvant utiliser plusieurs niveaux d'abstraction, plusieurs langages ainsi que des incompatibilités de protocoles de communication.
2. le partitionnement logiciel/matériel des fonctionnalités et communications de tous les nœuds de calcul et canaux de communication décrits au niveau *Transaction* afin d'en obtenir des spécifications au niveau *Message*.
3. la génération du modèle *Micro-Architecture* à partir de cette spécification composée de composants et canaux de niveaux *Message* ou *micro-architecture*, hétérogènes en langages de description et protocoles de communication.

3.5 Conclusion

Au long de ce chapitre, des outils intervenants à différentes étapes de la conception de systèmes monopuces ont été présentés. Étant donné l'étendue du flot de conception, ces outils n'ont qu'une action partielle et localisée dans le processus d'implémentation. Ces actions ont été présentées en deux familles :

1. la recherche macroscopique d'une solution d'implémentation : l'*exploration d'architecture*.

   Ces outils dont les caractéristiques sont synthétisées dans le tableau 3.1 sont soit inadaptés pour traiter des motifs architecturaux génériques (*Cosyma*, *Archimate*), soit trop découplés de l'implémentation pour permettre une utilisation automatisée de leurs résultats (*VCC*) ;

2. le raffinement de la solution architecturale macroscopique en modèles plus pragmatiques et suffisamment détaillés pour autoriser leur prise en charge par des flots classiques et automatiques de compilation logicielle ou de synthèse matérielle : l'*intégration de composants*.

   Les outils d'intégration peuvent, grâce aux critères rassemblés dans le tableau 3.3, être classés en trois catégories :

   (a) les outils capables de générer des modèles spécialisés des réseaux de communication (*N2C*, *Program*, *Gaut*). La principale limitation de ces outils réside dans leur capacité à n'adresser que des topologies (structures) fixées. En générale il s'agit de réseau en étoile ou connexions point-à-points ;

   (b) les méthodes et outils construisant l'architecture autour d'un modèle figé de réseau de communication (*Coral*, *μNetwork*, *VCI*). Le support d'un unique protocole et d'une topologie figé rend difficile la diffusion de ces méthodologies auprès des utilisateurs disposant d'un portfolio de composants non-compatibles ;

   (c) les outils permettant d'abstraire les réseau de communication et de manipuler leur modèles comme des composantes architecturales. A notre connaissance, seul *Roses* offre cette approche.

La contribution majeure de ces travaux au flot *Roses* adresse le raffinement automatisé d'un modèle virtuel d'architecture en un modèle d'implémentation. Cette automatisation est apportée par l'outil ASAG et ses bibliothèques que le chapitre suivant détaille.

	CoWare N2C	IBM Coral	O'Nils / Program	Gaut	Sonics	VCI	ROSES
Composants	Ports abstraits + *RPC*	Ports abstraits	ports RTL + grammaire régulière	ports *RTL*			port virtuel générique
Réseau de communication	point-à-point	Bus partagé et fixé	point-à-point		Bus μcommuté fixé	réseau générique	Composant générique
Adaptation des composants	Synthèse adaptateur	ϕ		Synthèse des *UCOM*	Synthèse manuelle		Synthèse du *Module Adaptor*
Synthèse des interconnexions	Synthèse de bus	Expansion et connexion directe	Synthèse manuelle	Interconnexion *RTL*		synthèse manuelle	Synthèse des *Chanel Adaptors*

TAB. 3.3 – Outils d'intégration de composants.

Chapitre 4

ASAG : un outil d'assemblage et ses modèles de représentation

Sommaire

Ce chapitre présente les détails du flot de génération d'architectures multiprocesseurs, hétérogènes et spécifiques aux applications tel qu'il a été développé au cours de cette thèse et présenté dans [53]. Ce flot s'articule autour d'un outil permettant une automatisation de la génération d'architectures suffisamment détaillées pour en autoriser l'implémentation. L'entrée de cet outil est une spécification abstraite de la structure de l'application reflétant les décisions macroscopiques d'allocation et d'assignation de ressources. Cette génération est exclusivement basée sur la sélection, la mise à l'échelle et la spécialisation de composants de bibliothèque permettant, grâce à un assemblage systématique d'obtenir la structure finale de l'architecture. La première section de ce chapitre présente les représentations de concepts architecturaux autorisant une manipulation systématique. La seconde détaille l'enchaînement des opérations opérées sur ces concepts.

4.1 Les représentations pour le fbt de génération

Plusieurs représentations de concepts architecturaux sont utilisés par la méthode de génération mise en place par cette thèse. Ces représentations permettent de modéliser 1º l'application réclamant des ressources de calcul et de communication, 2º la bibliothèque décrivant la structure et le comportement de chacune de ces ressources et 3º l'architecture résultante de la génération.

4.1.1 Organisation générale des représentations utilisées pour la génération d'architectures

La méthode de génération développée requiert trois représentations en entrée afin de pourvoir à une représentation de l'architecture en sortie. Ces quatre représentations, ainsi que leur utilisation sont illustrées par la figure 4.1.

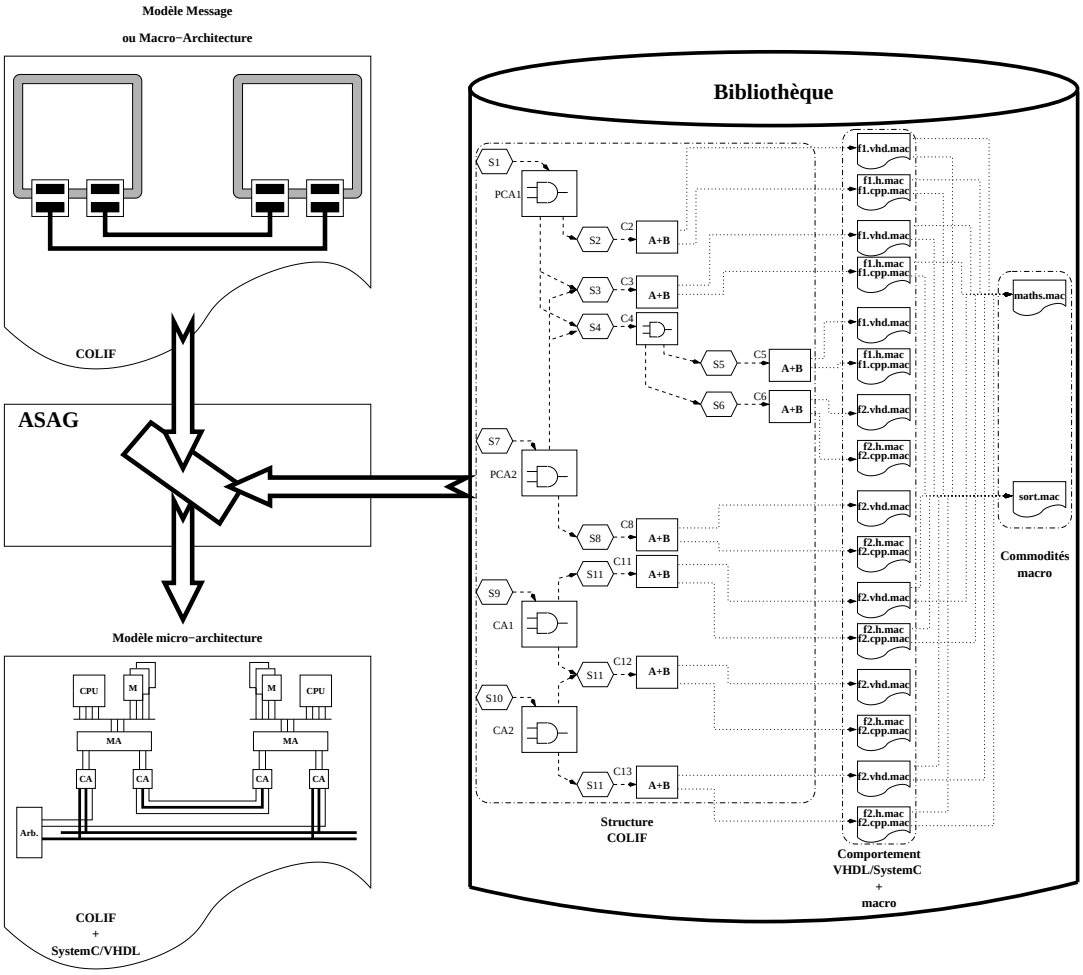


FIG. 4.1 – Représentations architecturales pour la génération d'architectures détaillées

Ces représentations sont :

1. la spécification initiale de l'application,
2. la description structurelle de ressources fournissant des services et disponibles en bibliothèque,
3. la modélisation générique de l'implémentation de ces ressources,
4. un modèle structurel de l'architecture générée et les modèles d'implémentation de chacun de ces composants.

Les spécifications des points 1, 2 et la description structurelle de 4 sont décrites en langage COLIF dont la construction est présentée dans la sous-section suivante. Les modèles du point 3 sont appelés macro modèles et seront détaillés en sous-section 4.1.4.

4.1.2 La bibliothèque architecturale

La bibliothèque de raffinement architecturale a pour responsabilité de fournir des implémentations de ressources répondant aux services réclamés par le modèle de l'application au niveau *Message*. Ces services sont de deux types : 1^o des services de calcul et 2^o des services de communication. Deux préoccupations fondamentales ont guidé la construction de cette bibliothèque, il s'agit de :

1. l'assemblage systématique ;
2. une couverture conséquente de l'espace de solutions.

Ceci est atteint par un découplage des ressources répondant à deux types de services précédemment décrits, autorisant ainsi leur composition orthogonale.

Concepts de base pour la bibliothèque

La bibliothèque répond aux besoins en calcul par l'offre de modèles d'Architecture Locale (AL) générique permettant l'implémentation sur mesure d'un type de nœud de calcul autonome. Quant aux services en communication, ils sont à la charge de 1^o des modèles spécialisables d'adaptateurs de canaux ou Chanel Adaptor (CA) génériques et compatibles à la fois avec les accès logiciel et les protocoles des ressources physiques de communication et 2^o de modèles extensibles de médias physiques de canaux de communication.

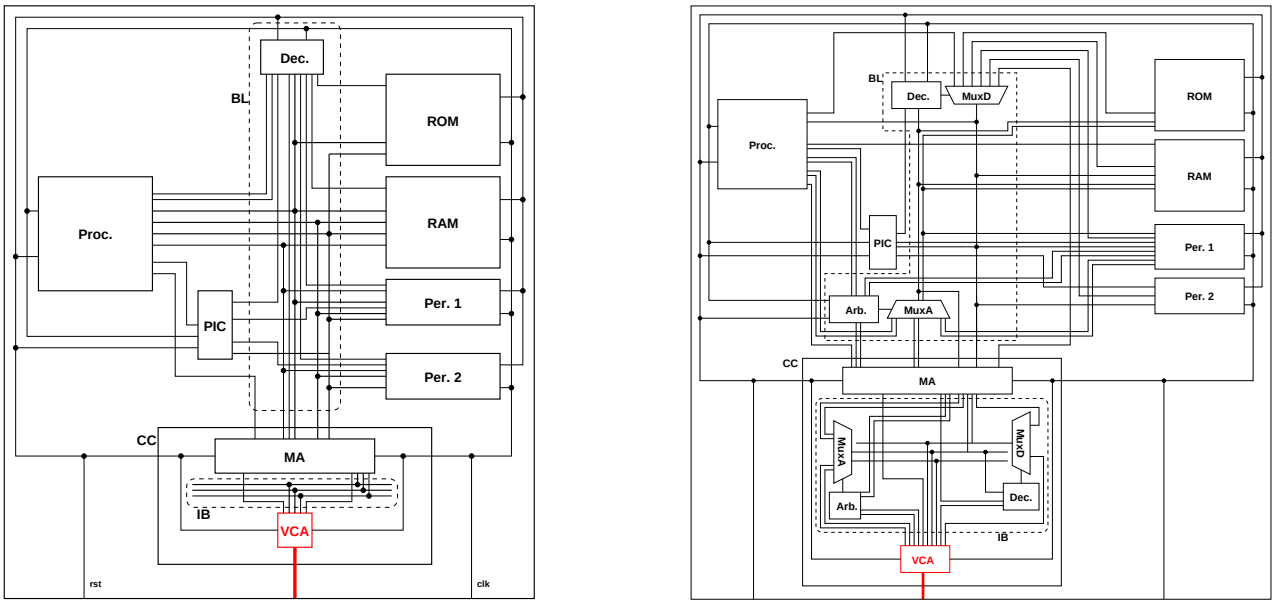
Détails sur les objets de la bibliothèque

Les modèles génériques d'AL peuvent être considérés comme des motifs d'architectures destinés à remplacer dans la structure de l'application les nœuds de calcul. Une sélection de ces modèles en fonction des services qu'ils fournissent, mais aussi une spécialisation et une mise à l'échelle de ces services doivent être opérées. Pour cela, une description structurelle des ressources (offertes par la bibliothèque) est insuffisante. Une sémantique particulière est donc adjointe à chaque composante de cette structure.

Motif d'architecture locale ou AL générique La structure générique d'une architecture locale peut naturellement être définie en COLIF, comme l'illustre la figure 4.2(a). Cet exemple de structure d'architecture locale se compose de :

- Une instance de processeur `Proc.` ;
- Un bus local `BL` ;
  - Les médias sont :
    - Des signaux `DOUT` et `DATA` de donnée ;
    - Un signal (bus) A d'adresse ;
    - Quelques signaux de contrôle (`M`, `nRW`, `nBW`, etc.) ;
  - Un simple décodeur d'adresse (`Dec.`), car seul le processeur se comporte comme maître dans cette architecture locale qui ne requiert donc pas d'arbitrage ;
  - L'interface est limitée à sa plus simple expression : un jeu de ports défini par le protocole du bus.

  Cette implémentation de la structure fonctionnelle reporte la résolution des signaux partagés tels que DATA au sein même des composants connectés et ce par l'utilisation de portes trois-états.
- Un bloc de mémoire `ROM` destiné à embarquer le code exécutable et les données constantes ;
- Un bloc de mémoire `RAM` (EDRAM simple port) pour contenir les variables de calculs ;
- Une unité matérielle de gestion de requêtes d'interruptions (multiplexage), le PIC ;



(a) bus mono-maîtres et portes trois-états.

(b) bus multi-maîtres multiplexés.

FIG. 4.2 – Motifs d'architectures locales.

– Des périphériques `Per.1` et `Per.2` tels que des coprocesseurs arithmétiques ou des temporisateurs (nécessaires à la gestion dynamique de l'exécution des tâches logicielles concurrentes confrontée à des contraintes de temps-réel).
– Un coprocesseur de communication `CC`, contenant :
  – Un adaptateur de module `MA`, spécifique au processeur `Proc.` ;
  – Un bus interne `IB`, ici implémenté par un bus partagé mono-maître. Son arbitrage est réalisé par un décodeur d'adresse embarqué dans le `MA` qui est le seul maître du bus. Les médias sont :
    – un signal (bus) de données bi-directionnel et partagé ;
    – des signaux d'activation des adaptateurs de canaux, générés par la logique de décodage d'adresse ;
    – des signaux de requêtes d'interruption émis par les adaptateurs de canaux en direction du maître, le `MA`.
  – Une instance unique d'adaptateur de canal appelé *Virtual Chanel-Adaptor (VCA)* car son interface n'est que partiellement définie alors que son comportement est inconnu. Cette instance permet 1° de spécifier la connexion de tous CA au sein du CC, tant du coté `IB` que vers l'extérieur (par l'usage d'un canal virtuel) et 2° de restreindre l'ensemble des CA présents en bibliothèque aux seuls compatibles au bus interne.

La figure 4.2(a) n'illustre qu'un exemple d'architecture générique locale très simple. En effet, les différents bus utilisés ne voient qu'un unique maître, aussi leur implémentation est des moins coûteuses. De plus l'interface de ces bus utilisent des signaux résolus par des portes trois-états. Enfin, l'utilisation de plusieurs unités de calcul au sein de l'AL est rendue possible par la spécification de plusieurs instances de processeurs connectés au même bus local (ex. : plusieurs processeurs ARM7TDMI connectés à un bus AHB AMBA). Afin de montrer l'espace de solution couvert par ces modèles génériques, un deuxième exemple présenté dans la figure 4.2(b) dépeint une implémentation qui représente actuellement la limite de complexité supportée par les modèles de représentation (Bibliothèques) et les mécanismes de génération (Algorithmes pour l'automatisation). Ainsi des bus multi-maîtres sont utilisables tant pour le bus mémoire locale que pour le bus interne au coprocesseur de communication. La résolution des signaux partagés tels que l'adresse, les données et les signaux de contrôle des accès est ici prise en charge par des multiplexeurs. Les multiplexeurs de données `MuxD` sont contrôlés par des décodeurs d'adresses, alors que les multiplexeurs d'adresses et de signaux de contrôle `MuxA` sont contrôlés par des arbitres.

Ces modèles structurels doivent permettre une extension de l'architecture afin que cette dernière réponde aux besoins de l'application. Cet objectif est atteint grâce à une identification du type de ces composants et de son assujettissement à une mise à l'échelle :

– L'identification du type des composants est codée par des objets de classes `MODULE_TYPE`, `NET_TYPE` et

PORT_TYPE.
– La nécessité de mettre un composant à l'échelle est codée par un PARAMETER attaché au composant : proliferation.

Les types de composants et leur sémantique sont expliqués dans le tableau suivant :

TAB. 4.1 – Types des composants de bibliothèque et sémantiques inhérentes.

PORT_TYPE	
valeur	Signification
CLK	Qualifie les ports d'horloge.
CONTROL	Caractérise les ports participant au contrôle des protocoles.
DATA	Ports aux travers desquels les données sont échangées.
ADDRESS	Ports emettant ou recevant l'index de la ressource ciblée.
DL	Port macro-architecture de communication, destiné à être remplacé par un ensemble de ports micro-architecturaux spécifiques à un protocole.

NET_TYPE	
valeur	Signification
clk	Signal d'horloge.
data	Nets transportant des données.
address	Signaux transportant des index.
int	Signal de requête d'interruption.
sel	Signal de sélection/activation d'un composant matériel.
ctrl	Signal de contrôle.
abschanel	Canal de communication macro-architectural destiné à être remplacé par un ensemble de signaux micro-architecturaux spécifiques au protocole.

MODULE_TYPE		
valeur	type	Signification
HARDWARE	CCPUxxxx	Qualifie les modules modélisant des cœurs de processeur de type xxxx
	CHpc	Qualifie les modules modélisant les CA.
	CHcpuspec	Caractérise les MA.
	CHdeco	Permet d'identifier un module comme décodeur d'adresses.
COMPOUND	CCarchi	Identifie les AL.
	CCconnect	Caractérise les modèles génériques de réseaux d'interconnexions.
	CCcc	Qualifie les coprocesseurs de communication.

La sémantique pour la mise à l'échelle est apportée par le paramètre proliferation, attaché aux PORT_DECL, MODULE_DECL et NET_DECL. Les valeurs possibles de ce paramètre sont :
– node indiquant que ce composant (port, net ou module) n'est pas sujet à la mise à l'échelle ;
– channel indiquant que le composant doit être répliqué afin de répondre à chaque canal de communication.

Modèles génériques d'adaptateurs de canaux Les CA sont modélisées en COLIF par des MODULEs de type CHpc. Leur interface est composée de deux parties, 1° un ensemble de ports permettant sa connexion au bus interne et 2° un port hiérarchique constitué de ports feuilles spécifiques au protocole *RTL* de l'implémentation du canal. Lors de la conception de ces modèles, deux philosophies ont été chronologiquement appliquées :
– Dans un premier temps, une modélisation monolithique de chacun de ces composants, implémentant une association statique des différentes couches fonctionnelles (introduites par le paragraphe 2.9.2). Seules les largeurs de données, les profondeurs des unités de mémorisation, les valeurs de priorités et adresses étaient alors configurables. Bien que de telles modélisations soient aisées à concevoir et mettre au point, elles ne couvrent qu'un espace restreint de solutions.
– Pour accroître la réutilisation de ces macro-modèles, une philosophie de modélisation à grain plus faible a été étudiée : l'implémentation de ces composants par l'assemblage d'implémentations configurés des différentes couches fonctionnelles les constituant. Ainsi, profitant de la symétrie des couches fonctionnelles des CA illustrées par la figure 2.20, des macro-modèles de leurs implémentations ont été conçus. Il en résulte une bibliothèque contenant trois familles de composantes : 1° les interfaces de lecture/écriture (« RWi »), 2° les transformations de données (« Processing ») et 3° les mémorisations internes (« Storage »). L'absence de la couche transverse de contrôle est remarquable et justifiée par son implémentation distribuée dans les autres couches. La configuration de

l'assemblage de ces couches et leur configuration propre permet la génération d'une à deux machines d'états RTL et un pool de données les séparant (le cas échéant). En effet, lorsque le CA n'est pas soumis à des accès internes et externes concurrents, une unique FSM suffit à contrôler l'ensemble. Mais si ces accès sont concurrents voire désynchronisés (synchronisés par des horloges différentes), alors deux FSM sont requises pour contrôler chaque coté de l'adaptateur découplé par la couche « Storage ». Le résultat le plus important est donc l'obtention d'un nombre minimal et suffisant de blocs matériels concurrents, et ce à partir de l'assemblage spécifique de 5 couches fonctionnelles présentées en section 2.9.2 et dans la figure 2.20.

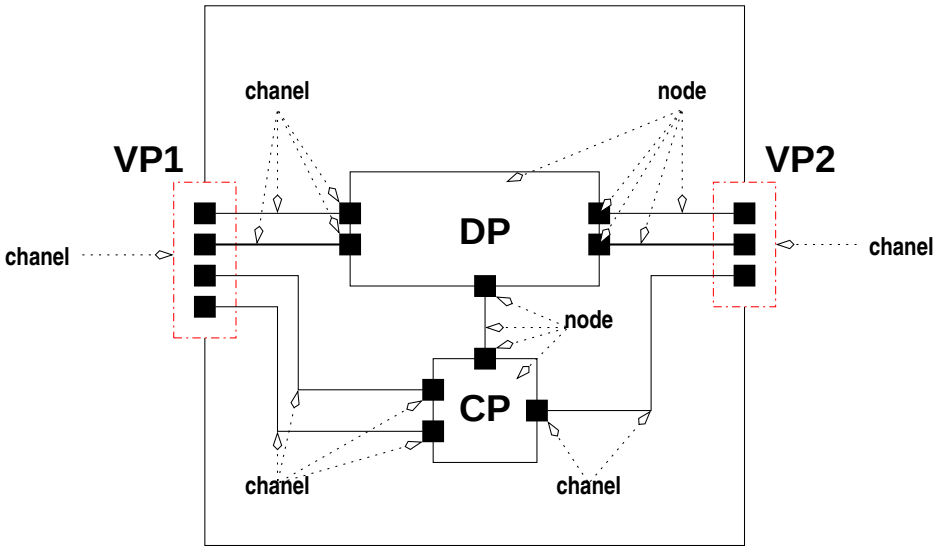


FIG. 4.3 – Exemple de modèle générique d'implémentation de canal de communication.

Modèles génériques de canaux Tout comme les motifs architecturaux d'AL, des motifs de réseaux peuvent être exprimés en utilisant un modèle générique annoté de paramètres *proliferation* en vue de guider sa mise à l'échelle. Un tel modèle est présenté par la figure 4.3. Il se décline en un module hiérarchique dont l'interface est constituée de ports hiérarchiques donnant les points d'accès au média supportés. Le média `DP` et sa logique de contrôle `CP` sont quant à eux modélisés par une structure de `MODULE_DECL`. Les objets devant être répliqués afin de mettre le média à l'échelle sont annotés du paramètre `proliferation` avec la valeur `chanel` alors que la valeur `node` annote les autres.

4.1.3 Spécification d'entrée du fbt de génération

La spécification d'entrée de ce flot est un modèle COLIF de l'application contenant des modules et canaux de communication virtuels (niveau d'abstraction *Message*). Les besoins en services et le dimensionnement des ressources à mettre en œuvre pour les implémenter sont matérialisés par des annotations de paramètres ou attributs de raffinement. Ceux-ci permettent de spécifier :

- L'architecture locale (ou AL) implémentant un module virtuel par `CPU`;
- Le type des données échangées par un canal (`DATA_TYPE`), ainsi que la taille de leur représentation (`DataBitWidth`);
- La taille du buffer interne au CA permettant de localement désynchroniser producteurs et consommateurs de données, peut être réglée par `POOL_SIZE`;
- Le protocole *RTL* du canal est spécifié par `HardPortType`;
- Des priorités statiques de requêtes d'interruptions émanant des CAs sont spécifiées par `CHAN_PRIO`;
- Lorsque qu'un réseau de communication ayant son propre plan mémoire est utilisé, alors `SHB_MASK`, `SHB_OFFSET` et `SHB_ADDRESS` permettent de configurer la translation des adresses entre le plan mémoire de local (AL) et le plan du réseau. De plus, `SHB_PRIO` permet de régler la priorité statique ou le nombre de tranches de temps allouées à l'accès d'un port au média.
- L'interface entre le logiciel et le matériel est spécifiée par les attributs suivants :

  `SoftPortType` permet de spécifier le pilote logiciel en charge d'interagir sur ce canal. Ce pilote est issu de [29].

`ADDRESS_DATA` renseigne le flot de conception logiciel et la génération d'architecture sur l'adresse physique allouée à un canal donné.

`ADDRESS_STATE` indique au flot de conception logiciel à quelle adresse physique est assigné le registre d'état d'un canal. Cette information est aussi utilisée pour la génération d'architecture, afin d'implémenter le comportement des décodeurs d'adresse et des arbitres de bus.

`MASK_GET` donne la position du drapeau de lecture, à l'intérieur du registre d'état d'un canal.

`MASK_PUT` est l'homologue de `MASK_GET`, puisqu'il concerne les opérations d'écriture.

`IT_NUMBER` spécifie l'identificateur de canal à retourner à la routine logicielle de traitement d'interruptions, lorsque cette dernière opère un accès en lecture à l'adresse `ADDRESS_INT` pour connaître le CA le plus prioritaire qui réclame une interruption.

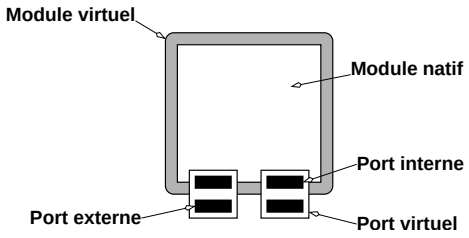


FIG. 4.4 – Localisation des attributs de raffinement.

Les objets annotables repris par la figure 4.4 sont le *module virtuel*, le *port interne* et le *port externe*. Ils représentent respectivement une encapsulation d'un modèle natif, une caractérisation des accès du comportement interne aux canaux de communication et une virtualisation de ces derniers. Leur définition a été abordée par la sous-section 3.4.1. L'attachement et la signification de ces attributs sont donnés par le tableau ci-après :

TAB. 4.2 – Attributs de raffinement.

`Attributs`	Attachement	Sémantique	Usage
`ADDRESS_INT`	Module virtuel	Adresse physique du gestionnaire de requêtes d'interruption	HW/SW
`CPU`	Module virtuel	Identifiant d'AL	HW/SW
`DATA_INT_WIDTH`	Module virtuel	Largeur du bus interne au CC	HW
`ADDRESS_DATA`	Port interne	Section pour l'accès au canal	HW/SW
`ADDRESS_STATE`	Port interne	Section pour l'accès au registre d'état du canal	HW/SW
`CHAN_NUM`	Port interne	Identifiant de canal pour la vecteurisation des requètes d'interruption	HW/SW
`MASK_GET`	Port interne	Position du drapeau de lecture	HW/SW
`MASK_PUT`	Port interne	Position du drapeau d'écriture	HW/SW
`SoftPortType`	Port interne	Identifiant du pilote	HW/SW
`DATA_TYPE`	Port interne	Type C++ des données	HW/SW
`IT_LEVEL`	Port interne	Priorité de l'irq	HW/SW
`DATA_BIT_WIDTH`	Port interne	Largeur native des données en interne au module.	HW
`DATA_BIT_WIDTH`	Port externe	Largeur du média externe inter-modules	HW
`CHAN_PRIO`	Port externe	Priorité relative des canaux pour les requêtes d'interruptions	HW
`HardPortType`	Port externe	Identifiant du protocole *RTL*	HW
`POOL_SIZE`	Port externe	Taille en nb de mots du buffer interne au CA	HW
`SHB_MASK`	Port externe	Masque pour la translation d'adresse entre le plan mémoire locale et le plan globale.	HW
`SHB_OFFSET`	Port externe	Offset de translation des adresses.	HW
`SHB_ADDRESS`	Port externe	Section de mémoire assignée à un port esclave.	HW

suite sur la page suivante.

suite de la page précédente.			
SHB_PRIO	Port externe	Priorité statique ou rapport cyclique pour l'accès à un bus partagé.	HW

4.1.4 Les «macro»-modèles de composants

Il s'agît d'un entrelacement de code en langage cible et de code en langage de macro externe permettant la génération paramètrée de code cible. Les macros, écrites dans le langage complet RIVE[29], génèrent les portions de code à configurer. Un exemple de macro-modèle apte à générer le code VHDL *RTL* pour un adaptateur de module spécifique au processeur ARM7 est donné par la figure 4.5. Cette figure est formée de la liste des contenus de trois fichiers :

1. un fichier décrivant un macro-modèle de MA spécifique au processeur ARM7TDMI et à un bus interne réduit à sa plus simple expression (signal de données partagé et signaux de sélection) est illustré par la sous-figure 4.5(a). Le macro-langage RIVE est ici très utile pour définir « dynamiquement » des ports, ce que le langage VHDL ne nous permet pas. Cette opération réalisée grâce à la macro-fonction décrite de la ligne 16 à la ligne 21 et à ses appels aux lignes 42, 43 et 44.
2. un exemple de fichier VHDL généré à partir de ce macro-modèle est présenté par la sous-figure 4.5(b). La définition itérative de ports peut être notée, elle prend place de la ligne 29 à la ligne 40.
3. les paramètres de génération utilisés sont donnés par le fichier listé en sous-figure 4.5(c). En particulier, le tableau `chan_name`, défini en ligne 6, permet de configurer le nombre et le nom des ports à définir.

```
@{# File : $BIBLIO/IB-0.0/LOCAL/ARM7TDMI/VHDRIV/ARM7MA.vhd.riv
# Purpose : Macro code for generation of ARM7-specific module
# adaptor behavior.
# Authors : AN & DL from static code by MB & XR
# Created on February 2K1 by AN from MacroExpander code by DL
}@-- File : @{node_name}@/ARM7MA_@{node_name}@.vhd
-- Module adaptor for ARM7TDMI CPU
-- Generated by ASAG

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.const_@{node_name}@.all;
use work.types_@{node_name}@.all;
use work.cust_@{node_name}@.all;
@{ DEFINE {prefix,dir,type,first,last}PORT_DECL =
      FOR port_num FROM (first) TO (last) DO
         ";\n\t"
         prefix"_"chan_array[port_num][0]"␣:␣"dir"␣"type
      ENDFOR
   ENDDEFINE

   DEFINE nb_chan = SIZEOF{chan_array} ENDDEFINE
}@

entity @{entity_name}@ is
  port (
   -- Processor specific ports
   CPIA      : in     T_address;
   CPIDATA   : in     T_data;
   CPIDOUT   : out    T_data;
   CPIMCLK   : in     std_logic;
   CPInBW    : in     std_logic;
   CPInIRQ   : out    std_logic;
   CPInM     : in     T_M;
   CPInRESET : in     std_logic;
   CPInRW    : in     std_logic;
   CPInWAIT  : out    std_logic;
   CPInCS    : in     std_logic;
   -- Internal Bus specific ports
   CPIdata_bus : inout  T_data@{
   PORT_DECL{"CPIen_n_data","out","std_logic",0,nb_chan-1}
   PORT_DECL{"CPIen_n_status","out","std_logic",0,nb_chan-1}
   PORT_DECL{"CPIevnti","in","std_logic",0,nb_chan-1}
   }@);

.
.
.

end @{entity_name}@ ;
```

(a) Macro-code RIVE.

```
-- File : VM1/ARM7MA_VM1.vhd
-- Module adaptor for ARM7TDMI CPU
-- Generated by ASAG

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.const_VM1.all;
use work.types_VM1.all;
use work.cust_VM1.all;


entity ARM7MA_VM1 is
  port (
   -- Processor specific ports
   CPIA      : in     T_address;
   CPIDATA   : in     T_data;
   CPIDOUT   : out    T_data;
   CPIMCLK   : in     std_logic;
   CPInBW    : in     std_logic;
   CPInIRQ   : out    std_logic;
   CPInM     : in     T_M;
   CPInRESET : in     std_logic;
   CPInRW    : in     std_logic;
   CPInWAIT  : out    std_logic;
   CPInCS    : in     std_logic;
   -- Internal Bus specific ports
   CPIdata_bus : inout  T_data;
   CPIen_n_data_p1 : out std_logic;
   CPIen_n_data_p2 : out std_logic;
   CPIen_n_data_p3 : out std_logic;
   CPIen_n_data_p4 : out std_logic;
   CPIen_n_status_p1 : out std_logic;
   CPIen_n_status_p2 : out std_logic;
   CPIen_n_status_p3 : out std_logic;
   CPIen_n_status_p4 : out std_logic;
   CPIevnti_p1 : in std_logic;
   CPIevnti_p2 : in std_logic;
   CPIevnti_p3 : in std_logic;
   CPIevnti_p4 : in std_logic);

.
.
.

end ARM7MA_VM1 ;
```

(b) Code VHDL généré.

```
# File ARM7MA_VM1.inc
# Generated by ASAG

DEFINE node_name = "VM1" ENDDEFINE
DEFINE entity_name = "ARM7MA_VM1" ENDDEFINE
DEFINE []chan_array = [ ["p1",0xF0018,0xF001C,32,1,1,5], ["p2",0xF0000,0xF0004,32,2,1,2]✂
   ➲, ["p3",0xF0008,0xF000C,32,3,1,3], ["p4",0xF0010,0xF0014,32,1,1,4] ] ENDDEFINE
```

(c) paramètres RIVE de génération.

FIG. 4.5 – Macro-modèle RIVE/VHDL→VHDL.

4.2 Raffinement des nœuds de calcul

Chaque nœud de calcul logiciel de la spécification *Message* du système est raffiné en une architecture locale. Des modèles génériques d'une telle architecture sont présentés dans la sous-section suivante. Puis une mécanique de construction d'architecture cible par assemblage systématique de composants est introduite.

4.2.1 Construction d'architecture locale

La génération d'une architecture locale à un nœud de calcul par mise à l'échelle d'une architecture générique revient à dupliquer des ressources. Le but étant de répondre au nombre de canaux de communication concurrents par un nombre adéquat d'adaptateurs de canal et de signaux, une méthode de modélisation topologique d'architectures génériques est introduite. Cette méthode doit permettre de spécifier simplement et naturellement la topologie et le caractère réplicatifs de composantes architecturales. La méthode établie se compose de 3 étapes :

1. Description d'une architecture locale avec comme hypothèse, l'utilisation d'un seul canal donc d'un seul adaptateur de canal. Cet adaptateur de canal utilisé uniquement pour la modélisation de l'architecture générique, n'est associé à aucun protocole, donc à aucun comportement. Il est désigné par le terme *Virtual Chanel-Adaptor (VCA)* précédemment introduit. Pour les mêmes raisons, son interface et sa connexion avec l'extérieur de l'architecture locale ne peuvent être que modélisées par un port et des canaux abstraits aux contenus vides.
2. Les modules, les ports et les signaux, devant être répliqués lors de la génération, sont identifiés par l'attribution d'un paramètre de description *proliferation* prenant alors la valeur «*chanel*». Le *VCA*, les signaux connectant ce dernier au **MA** de façon exclusive (point à point), les ports du **MA** utilisés pour ces connections, sont pour l'architecture de la figure 4.2(a) les seuls objets concernés par cette opération.
3. Toutes les ressources devant rester uniques après l'adaptation de l'architecture reçoivent le même attribut *proliferation* mais qui vaut cette fois-ci «*node*».

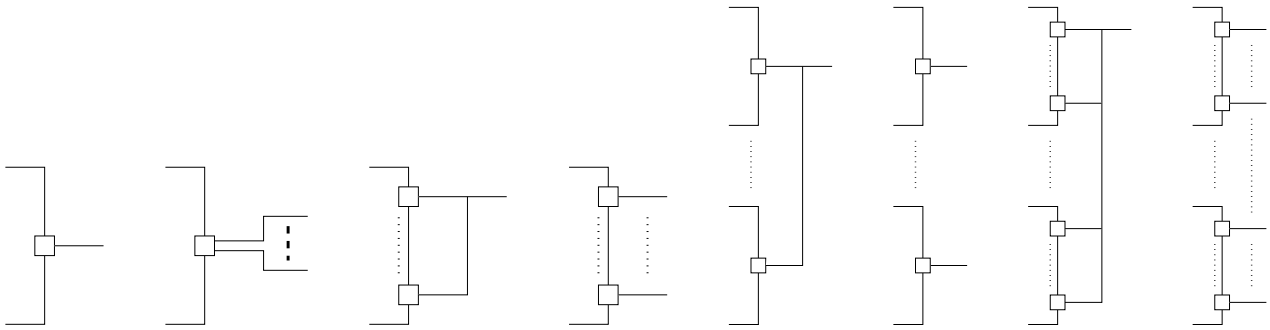


FIG. 4.6 – Modèle de réplication.

Modèle	Module	Port	Net
(a)	*node*	*node*	*node*
(b)	*node*	*node*	*chanel*
(c)	*node*	*chanel*	*node*
(d)	*node*	*chanel*	*chanel*
(e)	*chanel*	*node*	*node*
(f)	*chanel*	*node*	*chanel*
(g)	*chanel*	*chanel*	*node*
(h)	*chanel*	*chanel*	*chanel*

FIG. 4.7 – Combinaisons des attributs *proliferation*.

Mise à l'échelle

Interprétant ces informations de réplication l'algorithme de mise à l'échelle de l'architecture peut engendrer tous les modèles structurels présentés par la figure 4.6. A chacun de ces modèles correspond un modèle initial assigné d'un jeu d'attributs *proliferation*.
Cette correspondance est donnée par le tableau 4.7.

Spécialisation

La technique et la mécanique précédemment présentées n'adressent que la mise à l'échelle de l'architecture matérielle. Or certains de ces composants doivent être choisis parmi un ensemble. Cet état de fait concerne principalement les adaptateurs de canaux et les modèles d'architectures locales. Le choix des modèles d'architectures locales se fait en fonction des critères suivants :

1. Le support du processeur requis par les paramètres de raffinement du modèle de niveau *Message* de l'application ;
2. Le support de tous les adaptateurs de canaux mis en œuvre au sein du coprocesseur de communication de cette architecture locale ;
3. Le support de périphériques requis pour le bon fonctionnement de l'architecture locale. Ainsi le choix entre plusieurs architectures candidates sera orienté vers celles embarquant un temporisateur (générateur d'interruptions périodiques) lorsque plusieurs tâches logicielles concurrentes devront être exécutées sur le même processeur.

Quant au choix de l'adaptateur de canal, il s'opère sur l'ensemble des adaptateurs compatibles avec le bus interne de l'architecture locale précédemment sélectionnée. Il est dirigé par l'aptitude d'un adaptateur à contrôler le protocole du canal, spécifié par le paramètre *HardPortType*, et par sa compatibilité avec le pilote de communication logiciel spécifié par le paramètre *SoftPortType* et automatiquement généré par [29]. Ainsi ces paramètres permettent de déterminer l'implémentation d'un sous-ensemble inférieur de la pile protocolaire OSI. *SoftPortType* exprime le point d'entrée dans cette pile (permettant ainsi de déterminer la frontière entre l'implémentation logicielle et l'implémentation matérielle de la pile) alors que *HardPortType* spécifie les couches inférieures.
Un macro-modèle est associé à chaque CA à fin d'en générer des modèles taillés sur mesures et répondant au mieux aux besoins de l'application.

Couplage de la réplication et de la spécialisation

Ces opérations de réplication et de spécialisation sont exécutées conjointement.
Les adaptateurs de canal sont de bons exemples justifiant cette entrelacement. En effet, en plus de la décision de répliquer de tels composants, leur spécialisation doit être opérée. Ceci est encore plus flagrant, lorsqu'un adaptateur de canal est détourné de sa fonction première (adaptation des communications inter-modules) pour implémenter un service requis par un *SAP*. La connexion de tels adaptateurs avec l'extérieur de l'architecture locale n'est alors pas requise. Ces connexions ne sont donc pas mises en œuvre lorsqu'un port SAP est reconnu (par l'absence de port externe dans le *Port Virtuel*).
Nous avons décidé d'aborder cette complémentarité en séquentialisant la réplication et la spécialisation, de la façon suivante :

```
Pour M dans liste des instances de module du motif, faire :
  PVA = debut de liste des ports virtuels applicatifs
  faire :
    Selection : SelModule=Recherche(èBibliothque,PVA.fonctionnalites)
    Mise a l'echelle de l'interface : Pour P dans SelModule.interface
    faire :
      Si P.proliferation = chanel
      faire :
        Pour PVA2 dans liste des ports virtuels de l'application
        faire :
          Si PVA.type/=SAP ou P ne connecte pas M a la frontiere du module de hierarchie
               superieure, alors :
            dupliquer(P)
    Recursion hierarchique : application de cet algorithme a la structure de Selmodule;
```

```
    Configuration et generation : les modeles comportementaux de SelModule sont generes
        par expansion de macro-modeles
    IM = instance(SelModule)
    PVA = port suivant dans la liste des ports virtuels
  tant que M.proliferation = chanel et PVA /= fin de liste des ports virtuels
      applicatifs
  PVA = debut de liste des ports virtuels applicatifs
  faire :
    Pour N dans liste des instances de nets du motif, faire :
      Si PVA /= SAP ou N n'est pas connecte a la frontiere du module, faire :
        IN = instance(N)
        Pour P dans liste des ports connectes a N, faire :
          Connecter IN a tous les ports issus de la replication de P
    PVA = port suivant dans la liste des ports virtuels
  tant que N.proliferation = chanel et PVA /= fin de liste des ports virtuels
      applicatifs
```

4.3 Raffinement de réseaux d'interconnexion

La diversité des réseaux de communication variant de la simple connexion directe et point-à-point, jusqu'au réseau de topologie aussi complexe qu'irrégulière, leur formalisation n'est pas aisée et n'a pas été abordée par ces travaux. Cependant, les besoins en spécialisation lors de la génération de leurs modèles *micro-architectures* sont inexistants. En effet leur fonctionnalité qui n'est appelée à varier que dans de faibles proportions, rend suffisantes les caractérisations issues de la synthèse de communication. Celles-ci se traduisent en un choix type de réseau-protocole ne nécessitant alors plus qu'une mise à l'échelle.

4.3.1 Génération d'interconnexions

Cette étape de synthèse n'adresse que la mise à l'échelle de l'implémentation de chaque canal, et est similaire à la construction d'architecture locale précédemment abordée. Elle n'en diffère que par :

1. l'inutilité de l'étape de choix des fonctionnalités implémentées, car celles-ci sont implicitement données par les protocoles annotés dans le modèle architecturale d'entrée ;
2. la mise à l'échelle des interfaces, qui est ici contrainte par la nature des ports virtuels.

La figure 4.3 présente un exemple de modèle générique d'implémentation de canal de communication. L'annotation des différents éléments (instances de modules, de nets et de ports) permet de réguler leur réplication suivant l'algorithme suivant :

```
Pour PVA dans liste des ports virtuels de l'application relies par ce canal, faire :
   Pour M dans liste des instances de modules du motif, faire :
      Pour N dans liste des instances de nets du motif, faire :
      Pour PH dans liste des ports hierarchiques du motif, faire :
        si type(PVA.Port_externe) = PH faire :
          IPVA = copie(PH)
        si N connecte M a PH faire :
          si N.proliferation = chanel ou N non marque faire :
          IN = copie(N)
          marquer N
        si M.proliferation = chanel ou M non marque faire :
          IM = copie(M)
          marquer M
    Pour P dans M.interface faire :
      si P connecte a N faire :
        si P.proliferation = chanel ou P non marque faire :
          IP = copie(P)
          marquer P
          connecter(IP,IN)
Pour M dans liste des instances de modules du motif, faire :
   si M non marque, faire :
      appliquer algorithme de mise a l'echelle d'AL sur M
```

```
    sinon
        Pour P dans M.interface, faire:
    appliquer algorithme de mise a l'echelle d'AL sur P
Pour N dans liste des instances de nets du motif, faire :
    Si N non marque, faire :
        appliquez algorithme de mise a l'echelle d'AL sur N
```

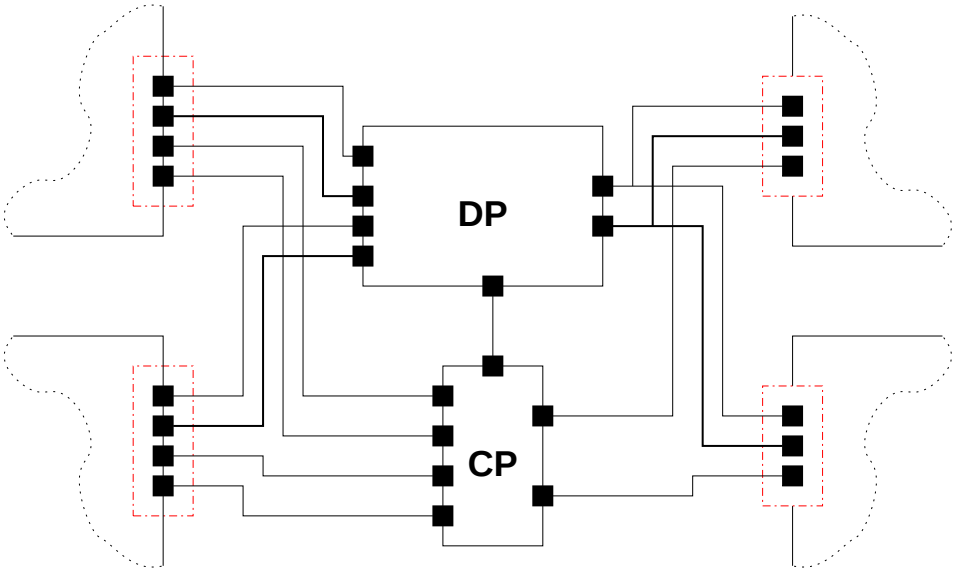


FIG. 4.8 – Canal raffiné par une implémentation spécialisée.

Ainsi, appliqué au motif de la figure 4.3, il est possible de générer le réseau illustré par la figure 4.8.

4.4 Éléments de bibliothèques

La bibliothèque développée pour ces travaux est conceptuellement présentée dans la sous-section 4.1.2. La voici maintenant décrite quantitativement.

4.4.1 Architectures locales d'un nœud de calcul

En termes d'AL générique, deux modèles sont disponibles :

1. Une architecture locale articulée autour d'une instance de processeur 68000 de Motorola ;
2. Une architecture à base d'une unique instance de processeur d'ARM7TDMI.

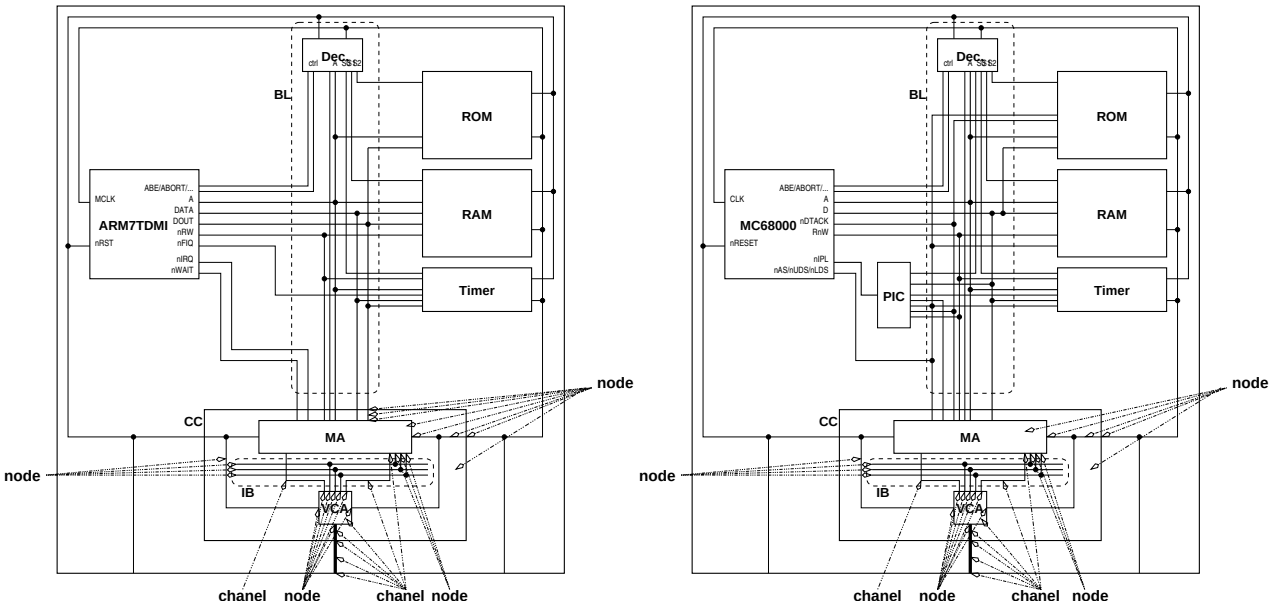
Ces modèles génériques d'AL sont illustrés par les figure 4.9(b) et 4.9(a). Toutes les composantes de ces deux architectures sont annotées du paramètre *proliferation*. La valeur par défaut de ce paramètre est *node* signifiant par là que la composante ne doit pas être répliquée. Parmi les composantes à répliquer, il est important de noter la présence des *VCA*, de leurs connexions à l'adaptateur de module, et des signaux les reliant aux frontières de l'architecture locale. Ce choix d'architecture locale est certes restreint mais il suffit à prouver la flexibilité de la méthodologie grâce à la variété des différences caractérisant ces deux processeurs. Parmi ces différences se trouvent :

– le mécanisme d'interruption à 7 niveaux de priorité, a source unique (`nIPL`) et auto-vecteurisation du 68000 contre la gestion de seulement 2 niveaux non vecteurisés associés à deux sources disjointes (`nFIQ` et `nIRQ`) de l'ARM7 ;
– un bus synchrone pipeliné pour l'ARM7 contre le bus asynchrone du 68000 ;
– des tailles de mots machines de 16 bits pour le Motorola contre 32 pour l'ARM.

Le support de ces spécificités est entièrement reporté dans les MA relatifs à chacune de ces architectures.

4.4.2 Adaptateurs de canaux de communication

Les AL présentes en bibliothèque utilisent toutes deux le même bus interne, l'offre en CA adresse donc des composants compatibles avec celui-ci. Il est ainsi possible de dénombrer 15 CA regroupés suivant les services qu'ils fournissent :



(a) Architecture locale générique à base d'ARM7.

(b) Architecture locale générique à base de 68000.

FIG. 4.9 – Motifs d'architectures locales à base d'ARM7 de 68000.

① Fifo asynchrone-synchrone : une paire de CA (un émetteur et un récepteur) est disponible afin d'échanger des données au travers d'une file. Le protocole utilisé pour interconnecter ces deux CA est une poignée de main à 4 phases permettant ainsi de découpler le séquencement (horloges) des terminaisons. La profondeur maximale de la file, la taille de chacune de ses cellules, la largeur de la connexion au bus interne, la largeur des données émises sont autant de grandeurs statiquement configurables. L'état de la file est consultable par scrutation. Pour permettre une synchronisation moins coûteuse avec des communications peu fréquentes, une requête d'interruption est aussi générée sur les changement d'états de la file (vide ↔ pleine).

② Fifo asynchrone : il s'agit d'une implémentation différente de la file de communication avec asynchronisme des terminaisons. Le découplage des régions d'horloge est ici réalisé par une mémoire double port offrant des performances et des coûts plus élevés que la solution asynchrone-synchrone (pas de poignée de main 4 phases). Les tailles de données et la taille de la file sont configurables.

③ Registre ou MMR : ces services (émetteur, récepteur) offrent une connexion directe à un registre de configuration ou d'états d'un composant direct. Aucune synchronisation n'est possible outre la comparaison en logiciel des résultats de deux lectures successives. De même cette connexion ne permet pas un découplage des régions d'horloge. L'inclusion au sein du CA d'un registre tampon, la largeur des données sont configurables.

④ Tampon : l'émission ou la réception (2 CA distincts) des données est morcelée par regroupement jusqu'à un seuil de déclenchement des données à échanger. Le seuil de déclenchement est configurable. Ce protocole n'offre ni synchronisation, ni l'asynchronisme.

⑤ Événement : les deux CA de ce service permettent l'émission/détection d'événement inter-nœuds de calcul, par le basculement d'un signal, qui après scrutation-comparaison matérielle est traduit en une requête d'interruption auprès du processeur de la terminaison réceptrice.

⑥ Double bloc mémoire commuté : deux blocs mémoires sont alternativement utilisés par les terminaisons communicantes. Chacune d'elles a la possibilité d'indiquer la fin d'utilisation du bloc qui lui est couramment assigné. Lorsque les deux blocs sont libérés, un mécanisme matériel de basculement permet d'interchanger les assignations terminaison-bloc. Une synchronisation est réalisée par le basculement des blocs (traduit en interruption) tout en permettant à chacun des nœuds de calcul de travailler avec sa propre horloge. La taille des deux blocs sont conjointement configurables.

⑦ AMBA AHB : deux CA embarquant des files permettent des connections Maître-émetteur et Esclave-récepteur à un bus AMBA AHB. Les tailles des données et des files sont configurables.

⑧ Service Access Port Timer : ce CA particulier offre un service de chronométrage. Il n'implémente pas une communication, mais permet de décompter un nombre de cycles horloges, préalablement et dynamiquement configuré par le processeur de l'AL. A la fin du décompte, une requête d'interruption est générée. Un prescaler peut être utilisé et statiquement fixé grâce à une sémantique détournée du paramètre `POOL_SIZE`.

4.4.3 Réseau d'interconnexion

Trois types de motifs sont disponibles :

1. les connexions point-à-point directes, où les ports sont directement connectés par couples ;
2. une connexion point-à-point bufferisée par FIFO asynchrone ;
3. un bus AMBA AHB, illustré par la figure 4.10. Les ressources (nets, ports, modules) devant être mises à l'échelle sont marquées d'un paramètre *prolifération* de valeur *chanel* (représentés par `C` sur la figure). Toutes les autres sont marquées d'un paramètre *prolifération* de valeur *node* (par défaut sur la figure 4.10).

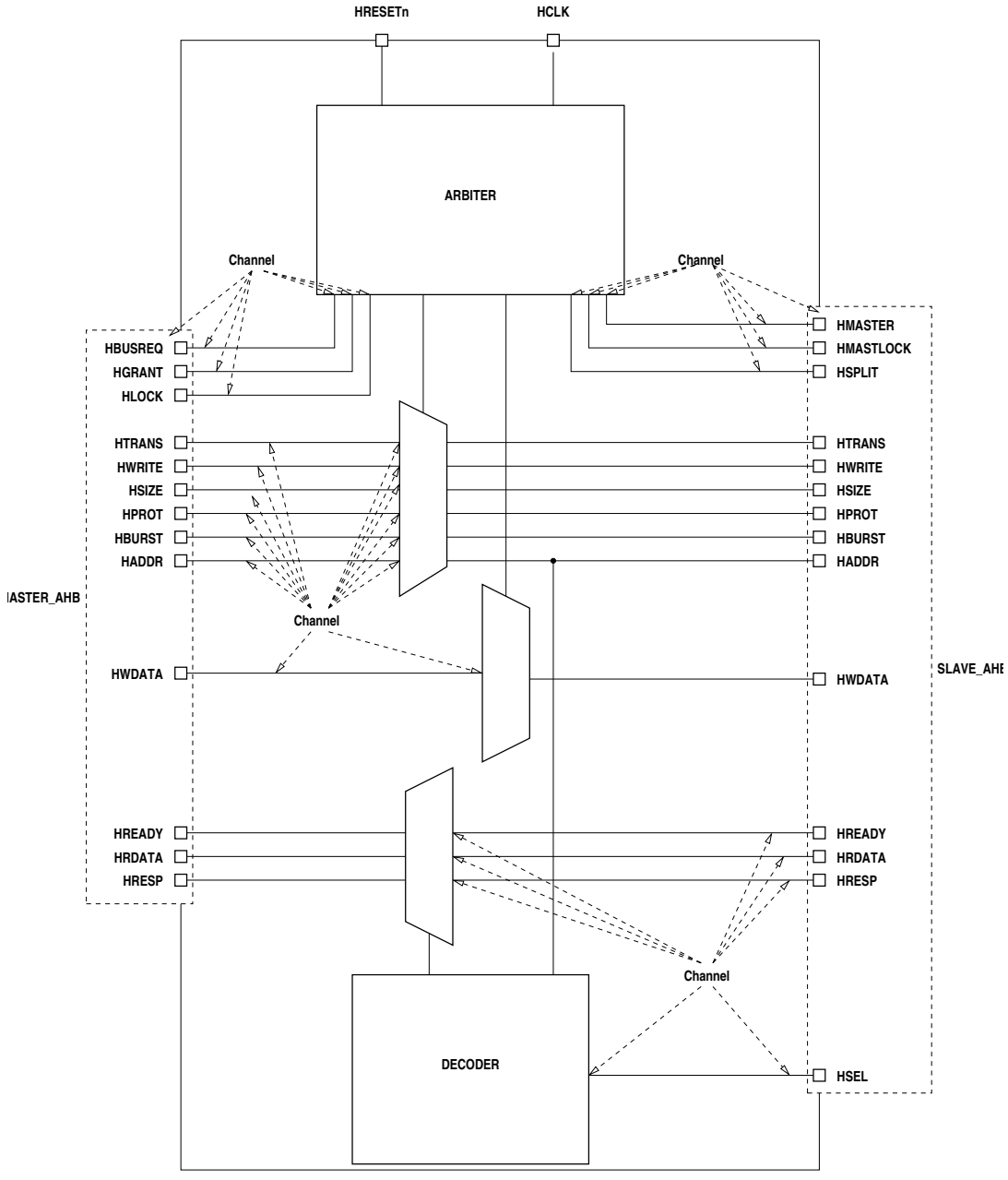


FIG. 4.10 – Motif d'interconnexion pour AHB.

4.5 Description du fbt de génération d'adaptateur matériel d'interfaces logiciel/matériel

La figure 4.11 représente la décomposition modulaire des fonctionnalités de l'outil de génération automatique d'architecture spécifique : **Application-Specific Architecture Generation (ASAG)**.

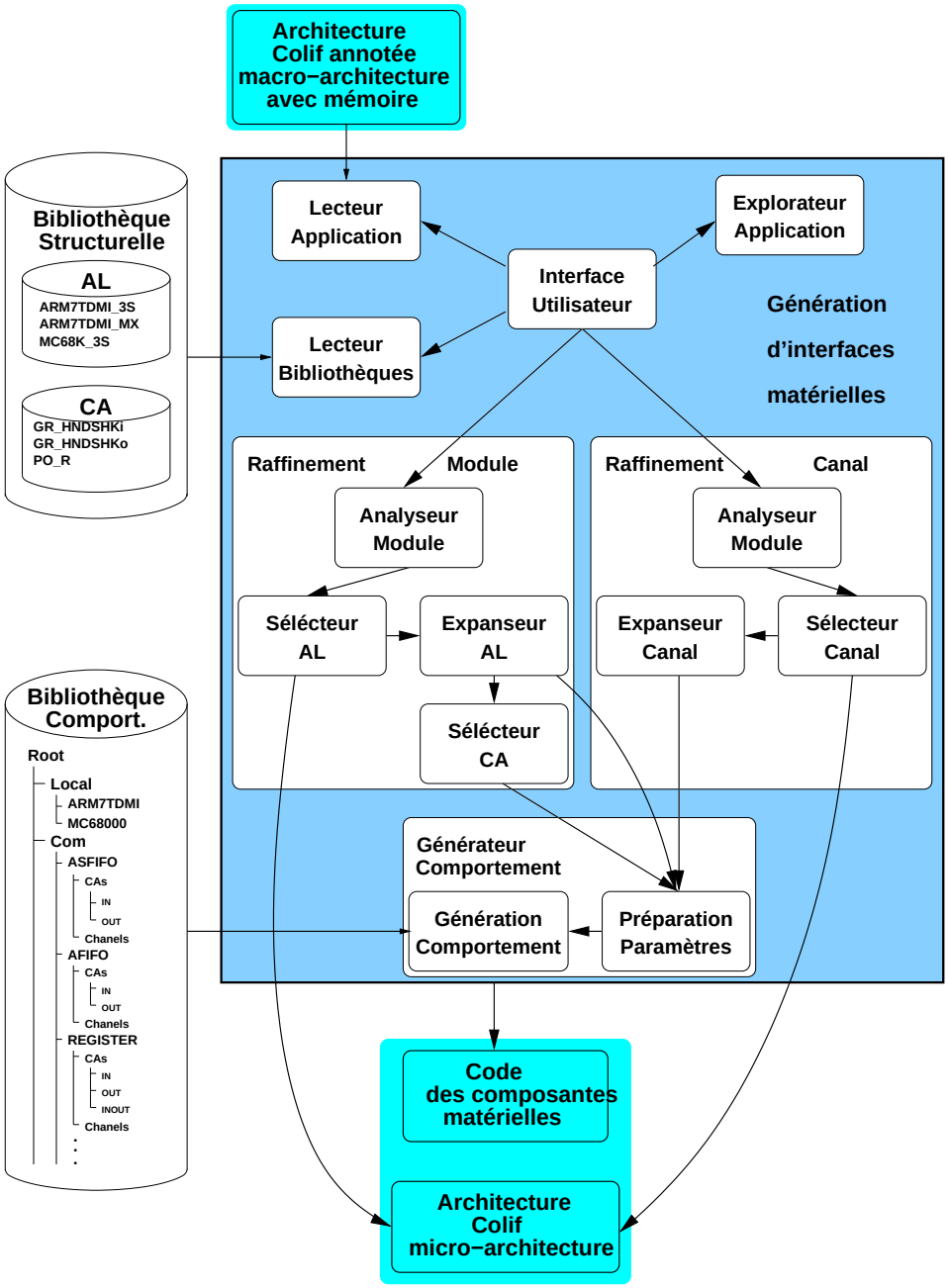


FIG. 4.11 – Architecture de la génération d'adaptateur matériel de communication.

L'architecture de cet outil se compose de 7 modules. Le premier module est dédié à la lecture et la vérification d'un modèle COLIF. Sa description est faite par la sous-section 4.5.1. Les bibliothèques de raffinement n'étant pas monolithiquement attachées à ASAG, un module de chargement dynamique, détaillé dans la sous-section 4.5.2 a été développé. La sous-section 4.5.3 présente l'opportunité de parcourir la structure du modèle de l'application afin de pouvoir appliquer localement un traitement. Le raffinement de module virtuel est présenté en détail par la sous-section 4.5.4. Les canaux virtuels sont raffinés par un module introduit par la sous-section 4.5.5. La mécanique de génération de code est développée comme un module indépendant illustré par la sous-section 4.5.6. Enfin, une couche d'interface

avec l'utilisateur est implémentée par un module décrit en sous-section4.5.7.

4.5.1 Lecture de la description de l'application

La description du système électronique à traiter, vient sous la forme d'un modèle COLIF tel l'exemple de la figure 3.16 présenté dans la sous-section 3.4.1. Techniquement, ce modèle est matérialisé par un fichier XML ayant pour définition de document le langage MARK-UP INTERNAL DATA-DESCRIPTION LANGUAGE EXTENSION (MIDDLE). Lors de cette étape les deux opérations suivantes sont chronologiquement opérées :

1. Analyse de la syntaxe MIDDLE et extraction de la définition de COLIF ;
2. Analyse de la syntaxe COLIF et construction d'un arbre contenant tous les objets définis (instances, déclarations, définitions de modules, nets et ports).

Pour des raisons évidentes de performances, deux optimisations sont utilisées. L'usage d'un analyseur de syntaxe XML associé au DTD MIDDLE est abandonné au profit d'un analyseur de syntaxe MIDDLE. En imposant la définition des structures de données COLIF en début de fichier XML, les deux opérations précédentes sont successivement réalisées en une unique lecture du fichier.

4.5.2 Chargement des bibliothèques

Les bibliothèques de génération d'architecture sont appréhendées comme des modèles COLIF adjoint d'une sémantique spécifique (cf. 4.4). En plus d'un arbre contenant la représentation de tous les composants utilisables lors de la construction d'architecture cible, ce module construit une base de données de ces composants classés par sémantique et compatibilité.
Cette base de données ayant pour but de faciliter les choix de composants, les trois dictionnaires suivant sont définis :

1. Un premier enregistrement est destiné à contenir la liste des modèles génériques d'AL, auxquels sont annotés le type de bus interne (IB), et le type de processeur utilisés ;
2. Un dictionnaire de CAs recensant pour chacun d'eux une liste de protocoles de communication contrôlables par le CA, une liste de pilotes logiciels de communication compatibles, une liste de bus internes compatibles ;
3. La dernière table est dédiée aux implémentations des canaux. Chacune de ses entrées est classée en fonction du protocole implémenté et contient une liste des ports abstraits de CAs compatibles.

4.5.3 Parcours de la structure du système

La hiérarchie du modèle *Message* en entrée n'étant pas contrainte, il est confortable de pouvoir afficher cette structure et indispensable de pouvoir se déplacer afin d'atteindre les nœuds de calculs à traiter.
Ces fonctionnalités sont implémentées par une manipulation des instances de modules COLIF semblable au parcours d'une arborescence de fichiers UNIX.

4.5.4 Raffinement de module virtuel

Le raffinement d'un nœud de calcul se décompose en l'analyse du module virtuel, la sélection d'un modèle générique d'architecture locale dans la bibliothèque, la mise à l'échelle de la structure de ce modèle générique et la spécialisation de chacun des adaptateurs de canaux.
Les quatre premiers paragraphes suivants tâcheront d'introduire ces étapes, alors que le cinquième présentera le résultat de cette opération.

Analyse du module virtuel

Afin de recenser les services de calcul et de communication requis par le nœud à raffiner, le module virtuel le modélisant est analysé. Les informations relatives au choix d'une AL et sa configuration sont récupérées parmi les attributs du module virtuel. Les choix de protocoles ainsi que les directions et largeur des canaux de communication sont récupérés lors d'un parcours des ports virtuels.

Sélection d'un modèle d'architecure locale

Un modèle générique d'architecture locale est sélectionné dans le dictionnaire d'AL en fonction de l'analyse précédente. Ce choix est basé sur la construction d'une liste d'AL candidates au raffinement et intégrant un processeur désigné par la valeur du paramètre *CPU*. Puis l'utilisateur doit élire l'AL désirée en se basant sur des informations de performances et technologiques (non fournies).

Expansion structurelle de l'architecture locale

Certaines composantes du modèle générique précédemment sélectionné sont répliquées pour permettre la mise à l'échelle de l'architecture locale.
Ainsi chaque unité assignée de l'attribut *proliferation* sera recopiée autant de fois que de canaux connectés à ce nœud de calcul.

Sélection d'un modèle d'adaptateur de canal

Lorsque l'étape d'expansion structurelle traite le VCA, elle active préalablement à la réplication de ce composant, le module de sélection de CA afin d'instancier un modèle spécialisé. Cette sélection dépend bien évidemment du protocole requis, mais aussi de la compatibilité du CA avec l'IB de l'architecture locale employée.

Module raffiné

La figure 4.12 illustre une AL obtenue par l'expansion et la spécialisation d'un modèle générique de bibliothèque.

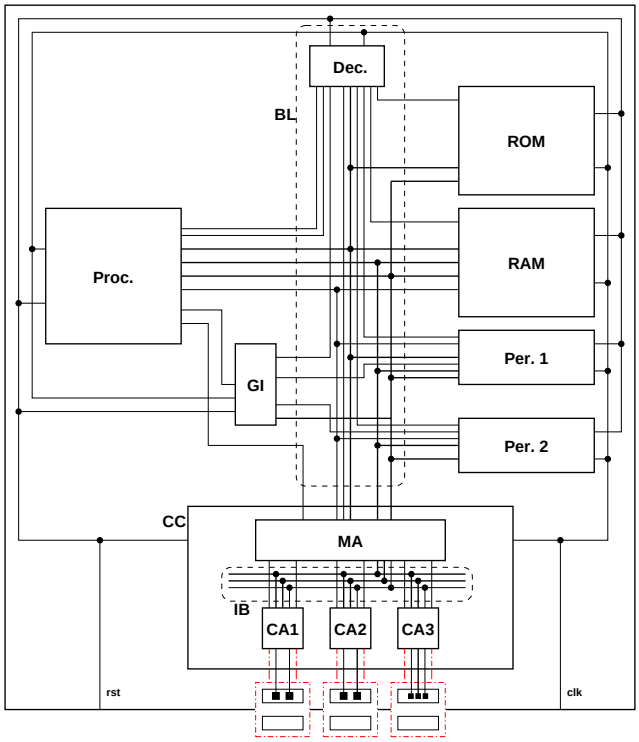


FIG. 4.12 – Exemple d'architecture locale générée.

Ce modèle final de l'architecture locale est composé de différents composants initialement présents dans le modèle générique (*Proc.*, *GI*, *Deco.*, *Per. 1* et *2*, *ROM*, *RAM*, *CC*, *MA*, *IB*) ou issus de la réplication d'une instance virtuelle (exemple : les CAs). Sa création et son insertion dans le modèle initial de l'application est le fruit d'une mutation de l'arbre d'objet COLIF créé par le module décrit en 4.5.1. Un fichier décrivant le comportement de chacun de ces composants (hormis le processeur et les mémoires) a été généré et est pointé par un lien inclus dans les modèles COLIF. Le raffinement partiel des ports de communication du nœud de calcul est notable. En effet ces ports sont hiérarchiques et contiennent :

Un **port interne** contenant les ports RTL spécifiques au protocole de communication ;

Un **port externe** issu du modèle du nœud avant raffinement.

Ce procédé permet de raffiner de façon modulaire les différents nœuds de calcul sans avoir à propager cette opération de nœud en nœud par le biais des canaux les interconnectant. Ainsi il est possible d'obtenir des modèles de l'application avec des modules et des canaux de communication de niveaux d'abstraction différents. Cette approche permet, moyennant l'adjonction d'interfaces de niveaux d'abstraction telles qu'elles sont présentées par [63], des gains notables concernant les performances de validation.

4.5.5 Raffinement de canal virtuel

Le raffinement de canal virtuel se décompose en trois parties : 1° l'analyse et recensement des caractéristiques du canal, 2° le choix d'une implémentation dans la bibliothèque, 3° la mise à l'échelle des composantes du canal. Chacune de ces étapes est détaillée dans les trois paragraphes suivants, alors que le quatrième présente le résultat de cette opération de raffinement.

Analyse du canal virtuel

Lors du raffinement d'un canal virtuel, un module d'analyse est mis en œuvre afin de parcourir et identifier tous les ports interconnectés par ce canal. La définition de ces ports, s'ils sont de niveau RTL, où leurs attributs (direction, protocole, largeur) permettent de construire une liste de caractéristiques guidant la sélection d'une implémentation de canal.

Sélection d'une implémentation du canal

En se basant sur la liste de caractéristiques précédemment construite, une implémentation de canal compatible est élue parmi les modèles recensés en 4.5.2.

Expansion du canal

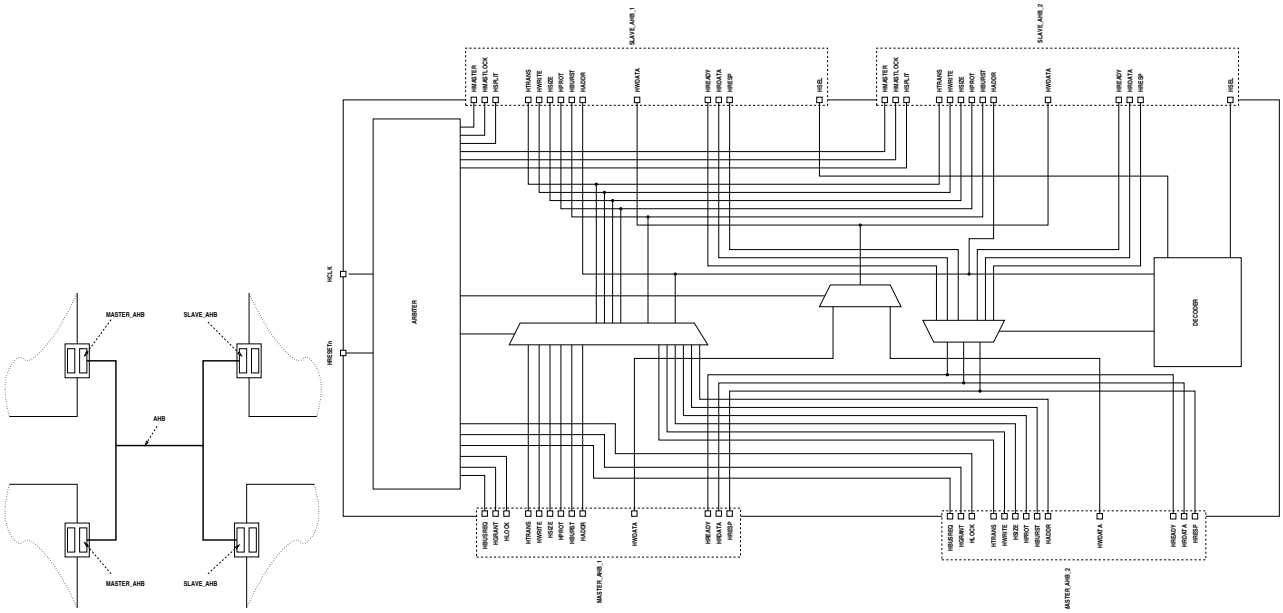
Afin de mettre à l'échelle l'implémentation choisie, une approche similaire à l'expansion d'AL est mise en œuvre. Cependant cette fois-ci, l'ordre de traitement des composantes de l'implémentation n'est pas indifférent. En effet, ici les ports connectés au canal sont des points d'entrée de l'algorithme de réplication. Partant d'un de ces ports, la réplication est appliquée à tous les nets, ports et modules connectés, si ceux-ci sont annotés d'un attribut *proliferation* ayant pour valeur *chanel*.

Canal raffiné

La figure 4.13(b) illustre un exemple de canal raffiné à partir du modèle générique présenté par la figure 4.10 et des spécifications illustrées par la figure 4.13(a).
Cette spécification réclamait l'utilisation d'un bus AHB, avec deux ports maîtres et deux ports esclaves (ces nombres sont arbitraires). Ce résultat est le fruit d'une exécution « manuelle » de l'algorithme de raffinement de canaux, son implémentation n'étant pas, à l'heure actuelle, suffisamment avancée pour supporter les connexions multipoints.

4.5.6 Expanseur de code

Ce module est en charge de la configuration et de la génération des modèles comportementaux des différents composants instanciés pour raffiner l'architecture du système traité. Il est donc activé par le module de raffinement de module.
Cette opération est, pour chaque instance de composant, constituée des trois étapes suivantes : 1° le recensement des groupes de macro-fichiers attachés au modèle générique du composant, 2° l'assignation de valeurs aux paramètres de configuration, 3° la génération des fichiers décrivant le comportement final.
Les trois paragraphes suivants sont consacrés à la description de ces étapes.



(a) Abstraction d'un bus AHB.

(b) Implémentation du bus AHB.

FIG. 4.13 – Raffinement d'un bus AHB.

Parcours de la liste de groupe de fichiers

Chaque modèle générique de composants en bibliothèque, hormis les processeurs et les mémoires, est annoté d'une liste de groupe de macro-fichiers. Ces macro-fichiers sont regroupés par type de macro-langage (RIVE, MacroExpander[1], M4, ...) utilisé et par type de langage cible (SYSTEMC, VHDL, ...).
La première étape de l'expansion de code est donc de recenser tous ces groupes, de créer pour chacun d'eux une arborescence (répertoire) spécifique.
Puis pour chacun des groupes de macro-fichiers recensés, un modèle d'outil effectivement capable de développer ces macro-fichiers en fichiers finaux est élu. Cette élection est basée sur l'association d'extensions de macro-fichiers à un modèle spécifique d'outil. En plus de cette liste d'extensions compatibles, ces modèles d'outils contiennent des méthodes utilisables par l'outil ASAG pour interagir avec ces outils.

Présentation des paramètres

Un fichier, définissant en macro-langage spécifique les paramètres de configuration, est généré pour chacun des groupes de macro-fichiers précédemment identifiés.
Ces définitions s'opèrent au travers d'appels de méthodes spécifiques, permettant la définition de paramètres atomiques ou de tableaux de paramètres.

Génération du code comportemental

Enfin, la méthode spécifique relative à l'appel de l'outil d'expansion est utilisée pour chaque macro-fichier. Les paramètres d'appel de cette méthode sont : 1^o le chemin du macro-fichier, 2^o le chemin du fichier de définition de paramètres et 3^o le chemin du fichier cible à générer.

4.5.7 Interface utilisateur

L'interface utilisateur de ASAG est basée sur un interpréteur de commande tel un shell UNIX.
Elles permettent :
– La lecture de fichiers COLIF des bibliothèques ou des modèles d'application ;

[1] Ancêtre de RIVE

– Le parcours de la hiérarchie du modèle d'application par des changements de niveau et des affichages de contenu ;
– Le raffinement indépendant de modules et de canaux ;
– La sauvegarde du modèle COLIF de l'architecture globale à tout instant, permettant une poursuite différée du raffinement.

4.6 Limitations et améliorations futures

Les objectifs de ces travaux ne sont pas l'obtention de résultats commercialisables, mais la mise en œuvre d'un prototype permettant l'obtention d'une maîtrise certaine de ce domaine de la conception. Il en résulte quelques limitations dont un grand nombre pourront être levées par de futures travaux.
Nous classifierons les limitations de cette approche de génération d'architectures spécifiques aux application en trois groupes : 1^o les limitations inhérentes à la méthodologie développée, 2^o celles relatives à l'implémentation de cette méthodologie par l'outil *ASAG* et enfin, les limitations des bibliothèques de ce dernier.

4.6.1 De la méthodologie

Quelques limitations méthodologiques qui s'apparentent plus à des hypothèses simplificatrices permettant de restreindre l'espace de solution, sont discernées :
– Les tâches sont statiquement assignées aux nœuds de calcul. Donc la migration d'une tâche d'un processeur à un autre n'est adressée que si l'AL assignée au raffinement de ce nœud est une grappe de processeur SMP et que la migration est confinée à l'intérieur du nœud de calcul. Cette limitation vient de la dépendance entre l'architecture matérielle et les groupes de tâches applicatives exhibés par la *Message*. Les architectures matérielles reconfigurables « en ligne » n'étant pas adressées par ces travaux, la résolution ou le contournement de ce point pourront être abordés dans le futur ;
– Le caractère « statique » des architectures matérielles imposant, comme nous l'avons précédemment mentionné, une topologie *Message* figée, les canaux et ports de communication de chaque module sont de topologie et de comportement statiques ;
– La complexité des réseaux de communication, mais surtout leur irrégularité et leur sensibilité aux paramètres technologiques poussent à l'utilisation de modèles prédéfinis ou IP appréhendés comme des modules par la méthodologie. Ici la limitation est relative à l'impossibilité de modéliser dans les modèles choisis, ces contraintes technologiques (propagation de signaux le long de lignes métalliques et description VHDL RTL) ;
– Le comportement de chaque tâche utilise des API de communication et de services devant être implémentable par des composants de bibliothèques. Il en résulte une relative contrainte pour les concepteurs lors de l'écriture de leurs modèles, mais aussi une confiance accrue en l'utilisation de composants bien définis. Les ensembles d'API et de leurs implémentations étant ouverts, les concepteurs peuvent espérer bénéficier d'un jeu de celles-ci répondant aussi adéquatement que possible aux spécificités du type de l'application développée.

4.6.2 De l'implémentation par l'outil

La matérialisation de la méthodologie en un outil d'aide à la conception apporte son propre lot de limitations. Si les aspects d'ergonomie et de confort d'utilisation sont mis de cotés, ces limitations sont :
– L'implémentation de l'algorithme de raffinement de réseau d'interconnexions ne permet actuellement pas la génération de réseaux de communication multipoints. L'impact sur l'utilisation de l'outil reste très limitée de par l'utilisation de générateurs propriétaires des modèles de réseaux de communication, permettant ainsi de les manipuler comme s'il s'agissait de propriétés intellectuelles. Cependant, quelques efforts de développement pourraient encore être portés pour générer des réseaux avec des motifs autres que des connexions point-à-point.
– L'interactivité entre ASAG et les macro-modèles est réduite à une transmission unidirectionnelle d'information : ASAG $\longrightarrow$ Macro-modèle. Ainsi les limitations d'un modèle comportemental détecté lors de la génération de ce dernier, ne peuvent être combattues par des modifications topologiques à posteriori. Le nombre maximum de connexions à un IB (réalisé par un bus partagé) peut nécessiter l'utilisation d'instances de bus supplémentaires. Cette décision doit actuellement être prise au préalable et être spécifiée dans le modèle d'entrée de l'application par le choix d'une AL supportant ce nombre de connexions.

– Le modèle de réplication basé sur la valeur statique de paramètres associés aux éléments de bibliothèque est candide. En outre, il n'offre pas la possibilité d'exprimer des irrégularités ou singularités topologiques. Par exemple, ceci rend impossible le chaînage de *Scan-Chain JTAG* entre les différents composants d'une *Micro-Architecture*. Le remplacement de cette interprétation d'un paramètre statique par l'offre d'une API de consultation aux composants de bibliothèques leur permettant de « s'auto-configurer » est en cours de développement.

4.6.3 Des bibliothèques

La principale limitation des bibliothèques d'*ASAG* est leur offre relativement pauvre en AL et en composants d'adaptation. En effet, on ne dénombre que :
– 2 AL, la première étant basée sur un processeur 68000 de Motorola et des bus (local et IB) partagés, alors que la seconde s'articule autour d'un processeur ARM7 et de bus partagés ;
– 15 adaptateurs de canaux pouvant être utilisés indifféremment dans l'une ou l'autre des AL.
– 6 réalisations de canaux, indépendantes des AL ;
– 1 SAP qui est en l'occurrence un temporisateur « chien de garde ».
La restriction à ces seuls composants est uniquement due aux ressources humaines engagées pour leur modélisation, la conception de leurs macro-modèles et leur validation-évaluation.

4.7 Conclusion

Dans ce chapitre, les modèles de représentation utilisés dans notre flot de génération, le raffinement des nœuds de calcul et de leurs communications ainsi que l'automatisation de ce raffinement par un outil ont été présentés. Puis les limitations relatives à la méthodologie mise en place, à l'état courant de maturité et de complétude l'outil et de ses bibliothèques ont été abordées. La validité de cet ensemble doit maintenant être prouvé par son utilisation sur différentes applications. Le chapitre suivant nous propose justement cette confrontation et une analyse des expérimentations.

Chapitre 5

Expérimentations

Sommaire

« Ce qu'il faut surtout, c'est monter beaucoup, tout en ne laissant pas les livres se couvrir de poussières sur les étagères. »

– Nuno OLIVEIRA.

Différentes applications de la méthodologie, de l'outil et de ses bibliothèques sur divers systèmes ont été opérées afin de valider cette approche, mais aussi d'évaluer la qualité des modèles générés. L'ensemble des applications traitées se compose d'un groupe de faible complexité (un anneau de tâches s'échangeant des jetons, une modélisation d'un établissement de restauration rapide), un groupe de complexité moyenne (un modem GSM WCDMA, un modem GSM IS-95, un routeur de paquets) et un système de complexité conséquente (un modem VDSL). Ce chapitre n'abordera que les études de cas relatives au modem GSM WCDMA et au modem VDSL. Le WCDMA nous a permit de valider la modélisation sous forme de réseau de modules virtuels ainsi que la génération d'un modèle RTL en SystemC destiné à la validation par co-simulation. Le modem VDSL, abordé dans la seconde section, nous a offert une validation plus poussée de notre flot de conception. Son modèle d'entrée fut raffiné jusqu'à un modèle exécutable sur plate-forme de prototypage à base d'architectures reconfigurables. Enfin une évaluation des résultats obtenus et une présentation des perspectives seront conduites.

5.1 Une station mobile GSM : WCDMA

5.1.1 Présentation du WCDMA

En Europe, la norme de communication des téléphones portables est le GSM. Le faible débit de cette norme limite l'utilisation de ces téléphones aux communications vocales et à un accès restreint à l'internet.
A partir de l'année 2002 des bandes de fréquences vont être attribuées à la nouvelle norme Wide Code Division Multiple Access (WCDMA). Grâce à son débit plus important[1], une nouvelle génération de téléphones portables va se développer. Ceux-ci deviendront de véritables terminaux multimédia miniatures qui permettront de consulter ses courriers électroniques, de télécharger des fichiers depuis l'internet.
La réalisation de cette nouvelle génération de téléphones nécessite d'intégrer des éléments et des protocoles de communication hétérogènes dans un volume restreint. De plus la conception de ces circuits doit respecter de fortes contraintes de performances, de coûts et de temps de réalisation. Pour satisfaire ces contraintes, on a recours aux systèmes monopuces qui permettent de réduire l'espace utilisé en intégrant sur une même puce plusieurs processeurs, DSP. Cette technique permet de réutiliser des composants standards réduisant ainsi les coûts et les délais de production. Par contre la diversité des composants constituant ces puces rend le réseau de communication très hétérogène. L'hétérogénéité du réseau rend sa réalisation très complexe.
Le but de cette expérimentation est d'appliquer ce flot de conception à une architecture simplifiée d'un décodeur WCDMA. Les outils du flot étant en cours de développement, ce travail consiste à définir une spécification de l'application et à compléter les outils de génération automatique.
Pour cela, le décodeur WCDMA sera dans un premier temps présenté, l'application du flot de conception du groupe SLS sera ensuite détaillée. Pour finir, quelques résultats et leur analyse seront livrés.

Décodeur WCDMA initial

Cadre de l'application Un modem WCDMA [76, 75] permet à un téléphone portable et à une station de base de communiquer. Ce modem se décompose en deux parties (figure 5.1). Une pour les communications allant de l'utilisateur vers la station de base (liaison montante) et une dans le sens inverse (liaison descendante).
La liaison montante se divise en deux étapes :

1. Le codeur code au format du modulateur des données qui proviennent des applications de l'utilisateur ;
2. Le modulateur convertit ce code en ondes électromagnétiques qui sont récupérées par la station de base de l'opérateur téléphonique ;

De façon symétrique, la liaison descendante s'effectue en deux étapes :

1. Le démodulateur convertit au format du décodeur les ondes électromagnétiques émises par la station de base ;
2. Le décodeur décode les informations provenant du démodulateur puis les transmet aux applications auxquelles elles sont destinées.

Dans ce chapitre nous allons nous limiter à l'étude du décodeur.

[1] 2 Mbps en situation fixe et 384 Kbps en mouvement.

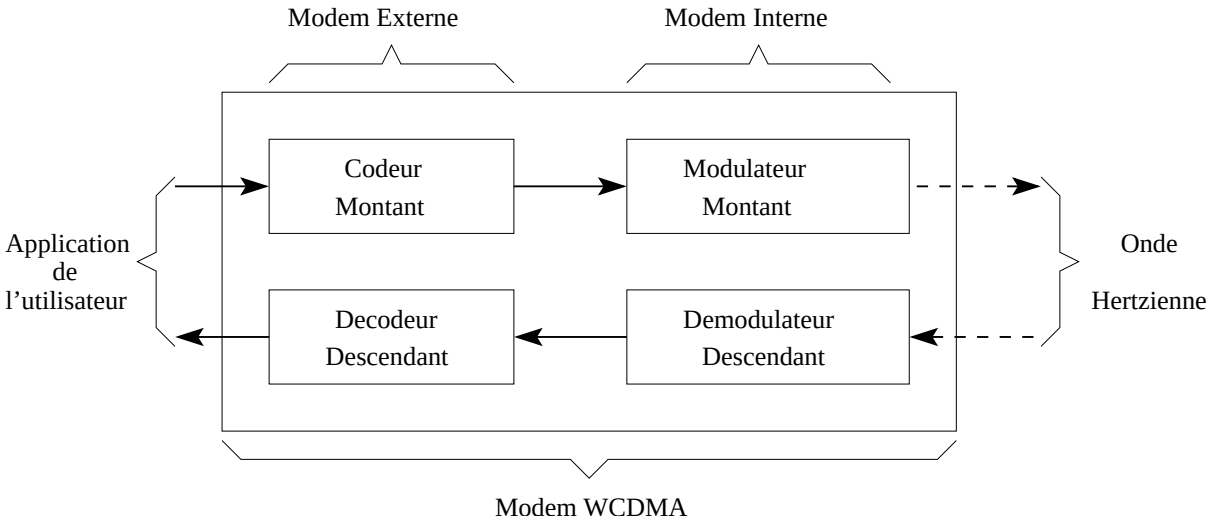


FIG. 5.1 – Structure d'un modem WCDMA

Architecture d'un décodeur WCDMA

L'architecture du décodeur (figure 5.2) est prévue pour assurer le traitement d'un flot de données. Ce traitement est assuré par un « pipeline » d'éléments matériels de calcul et un micro-contrôleur. Les entrées et sorties de données sont effectuées de manière asynchrone grâce à deux mémoires FIFO. Un banc de registre de configuration permet au micro-contrôleur de configurer les éléments de calcul.

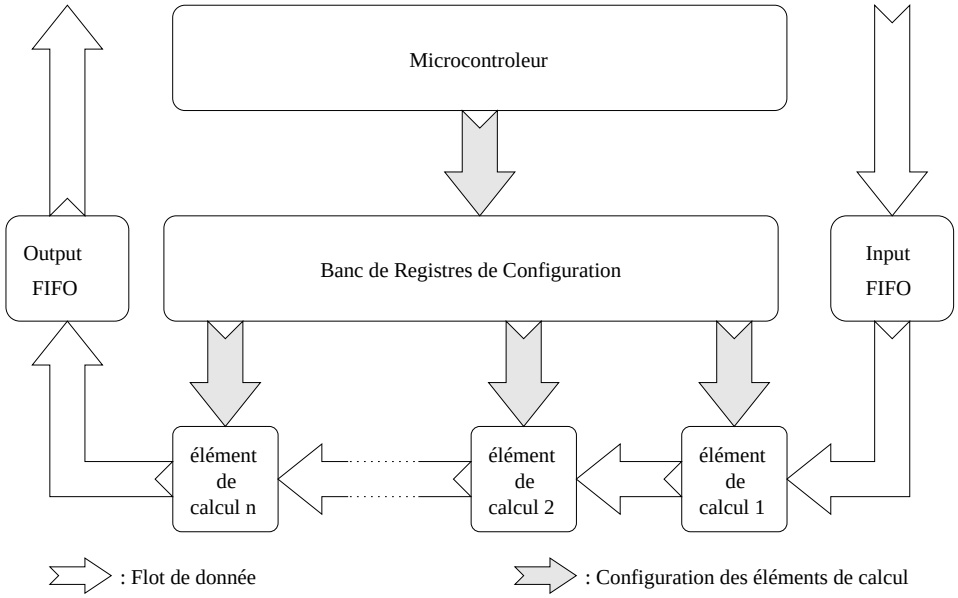


FIG. 5.2 – Modèle d'architecture d'un décodeur WCDMA.

Le flot de données est en fait divisé en paquets de taille variable. Le micro-contrôleur utilise les registres de configuration pour indiquer aux éléments du pipeline la taille du paquet, le type de calcul à effectuer et pour connaître l'état d'avancement du traitement du paquet.

5.1.2 Architecture du décodeur réalisée

Le groupe SLS ne possédant pas tous les composants du système, des simplifications de leur comportement ont été effectuées :

– pas de gestion d'énergie ;
– diminution de la taille du pipeline ;

– simplification des éléments de calculs :
  – pas de « bypass » ;
  – simplification des calculs effectués ;
– simplification des tâches à effectuer par le micro-contrôleur.

Malgré ces simplifications les résultats obtenus sont pertinents. En effet avec le flot de conception du groupe SLS, les composants ne sont que des briques interchangeables. Les simplifications effectuées ne changent pas les protocoles de communication de l'application initiale. On pourrait donc facilement remplacer les composants utilisés par les originaux.

L'architecture du décodeur (figure 5.3) se décompose en trois parties :
– les éléments de calcul ;
– les éléments de communication ;
– les éléments de contrôle.

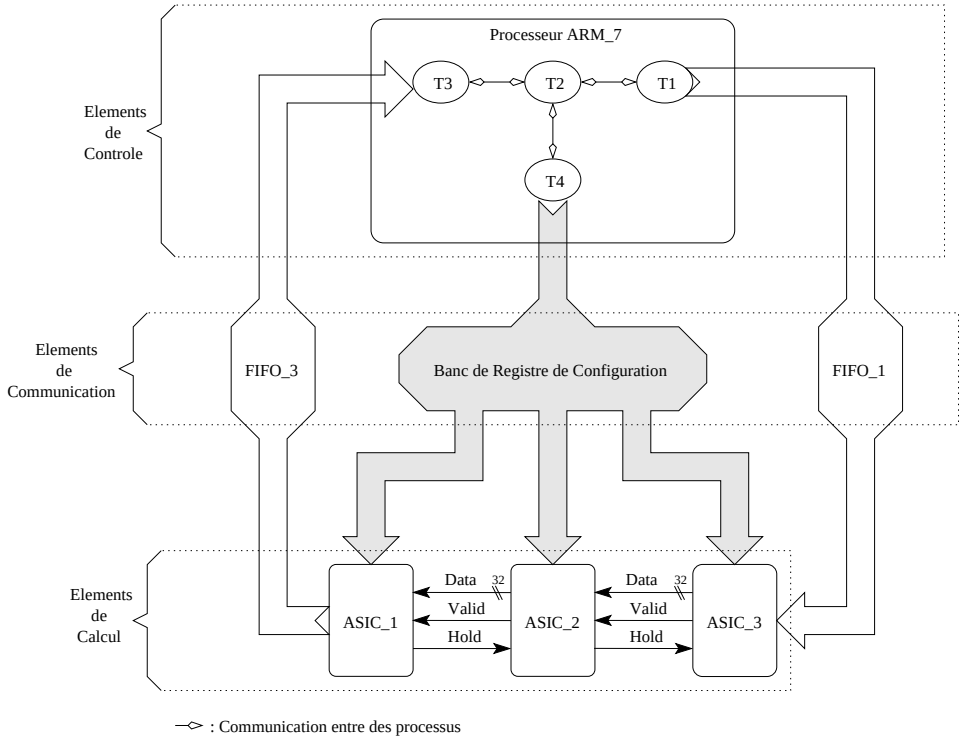


FIG. 5.3 – Décodeur WCDMA réalisé.

Éléments de calcul

Le calcul est réalisé par un « pipeline » qui se décompose en trois éléments de calcul ASIC_1, ASIC_2, ASIC_3. Ils possèdent chacun deux configurations et des délais de calcul différents (tableau 5.1).

Élément de calcul	Configuration	Calcul réalisé	Délais
ASIC_1	1	data + 1	1
	2	data * 2	4
ASIC_2	1	data + 2	2
	2	data * 3	6
ASIC_3	1	data + 3	3
	2	data * 4	8

(Délais en cycles horloges.)

TAB. 5.1 – Configuration des éléments de calcul.

La communication entre les composants du pipeline (figure 5.3) est synchronisée par deux signaux et un bus :

Data sert à échanger des données entre deux composants ;

Valid indique au consommateur que le producteur a mis une donnée sur le bus ;

Hold avertit le producteur que le consommateur est occupé et qu'il doit donc attendre pour envoyer une donnée.

Éléments de communication

FIFO Une FIFO est une mémoire à double accès séquentielle (sans adresse). Elle joue le rôle d'une file d'attente où les données sont lues dans le même ordre que celui de leur écriture. L'accès à cette FIFO est assuré grâce aux signaux (figure 5.4) :

Full indique si la FIFO est pleine ou non ;

Write signifie que la FIFO est accédée en écriture ;

Data_in est le bus qui sert pour écrire des données dans la FIFO ;

Read signifie que la FIFO est accédée en lecture ;

Empty indique si la FIFO est vide ou non ;

Data_out est le bus qui sert pour lire des données de la FIFO.

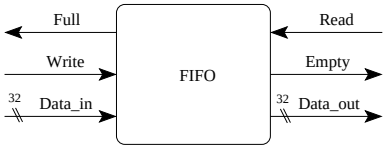


FIG. 5.4 – Signaux de la FIFO

Banc de Registres de Configuration Le banc de registres de configuration (figure 5.5) est implémenté comme un double banc de registres commutables. Le principe est de disposer de deux registres pour chaque canal de configuration. Ceci permet au micro-contrôleur d'écrire la prochaine configuration d'un élément de calcul en même temps que celui-ci lit sa configuration actuelle.
Ce banc de registres est accédé par le micro-contrôleur de façon identique à une mémoire traditionnelle grâce aux signaux :

Adresse détermine le registre accédé en écriture par le micro-contrôleur ;

Write signale que le banc est accédé en écriture ;

Data est le bus qui sert pour écrire des données.

Le seul changement est l'ajout de deux signaux pour gérer la commutation du banc de registres :

Switch demande de commuter le banc de registres ;

Write avertit le micro-contrôleur que tous les éléments du « pipeline » ont fini leur travail.

Pour l'élément **1** du « pipeline », l'accès à son registre de configuration s'effectue grâce au signal **Conf_1**. De même pour les ASIC_2 et ASIC_3 avec les signaux **Conf_2** et **Conf_3**.
Le banc de registres de configuration permet aussi au micro-processeur ARM7 de connaître l'état d'avancement du traitement sur le paquet courant. Lorsque l'élément de calcul nř1 a fini d'effectuer son traitement sur le paquet courant , il se sert du signal **Interrupt_1** pour émettre une interruption vers le banc de registres de configuration. De même pour les ASIC_2 et ASIC_3 avec les signaux **Interrupt_2** et **Interrupt_3**. Ce banc utilise le signal **Interrupt** pour transformer toutes ces interruptions en une seule par un simple « ou logique ».

Éléments de contrôle

Le contrôle du système est assuré par un microprocesseur ARM7 qui appartient à la famille des processeurs RISC. C'est un processeur 32 bits, de basse puissance qui est couramment utilisé dans les SoC.
L'architecture logicielle de ce contrôleur (figure 5.3) de communication est assurée par quatre tâches et un système d'exploitation :

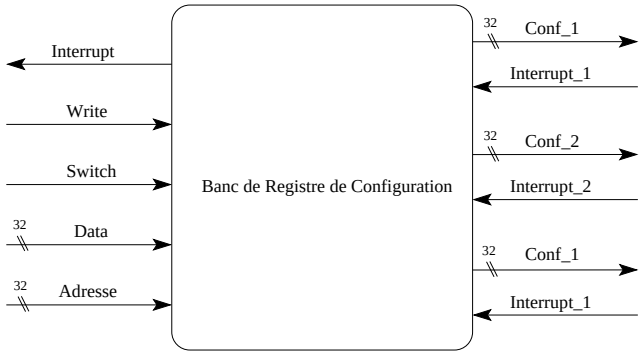


FIG. 5.5 – Signaux du banc de registres de configuration

– la tâche **T1** écrit des données dans FIFO_1 ;
– la tâche **T3** récupère les résultats du calcul du « pipeline » et vérifie leur correction ;
– la tâche **T4** écrit dans le banc de registre les configurations des différents éléments du « pipeline » ;
– la tâche **T2** coordonne les trois autres tâches en leur fournissant les données qu'elles doivent écrire ou tester.

Pour effectuer des opérations d'entrée/sortie, de communication, de synchronisation, ... ces tâches utilisent des API[2] fournis par le système d'exploitation.

5.1.3 Spécification du décodeur WCDMA

Deux spécifications différentes de l'application ont été réalisées.
Les deux sont découpées en deux modules de niveaux d'abstraction différents :
– Un module de niveau *Message* modélisant la partie de contrôle ;
– Un module de niveau *RTL* modélisant le « pipeline ».

La différence entre ces deux spécifications vient de la modélisation du banc de registre de configuration qui est :
– Dans la première spécification, une interface de communication que le flot doit générer ;
– Dans la deuxième, un élément de communication intégré au « pipeline ».

Cette différence montrera l'importance de l'étape de spécification et de la flexibilité du flot.

Première Spécification

Ce système (figure 5.6) est décrit avec deux modules :
– Le Module_ARM7 spécifié au niveau *Message* modélise la partie de contrôle du système. Il contient le comportement des quatre tâches logicielles T1, T2, T3, T4 ;
– Le Module_ASIC spécifié au niveau *RTL* modélise le « pipeline ». Il contient les trois éléments de calcul ASIC_1, ASIC_2, ASIC_3.

Le Module_ARM7, pour communiquer avec le pipeline, utilise trois ports virtuels qui servent de convertisseurs de niveau d'abstraction. Ils convertissent les primitives de communication de niveau *Message* de leur port interne en primitives de communication de niveau *RTL* de leurs ports externes.
Le Module_ASIC utilise cinq ports virtuels qui transmettent les primitives de communication entre leurs ports internes et externes sans les convertir.
La communication entre les deux modules s'effectue par des nets *RTL* regroupés dans cinq canaux hiérarchiques :
– Channel_Hierarchique_FIFO_1 permet d'accéder aux informations envoyées par le microprocesseur ARM7 ;
– Channel_Hierarchique_FIFO_3 permet au processeur de récupérer les résultats des calculs du « pipeline » ;
– Les trois canaux Ch_Hier_Reg_Conf_1, Ch_Hier_Reg_Conf_2 et Ch_Hier_Reg_Conf_3 permettent aux éléments de calcul d'accéder à leur configuration de calcul et d'avertir le processeur de la fin de leur travail.

[2] Des appels système dans notre cas.

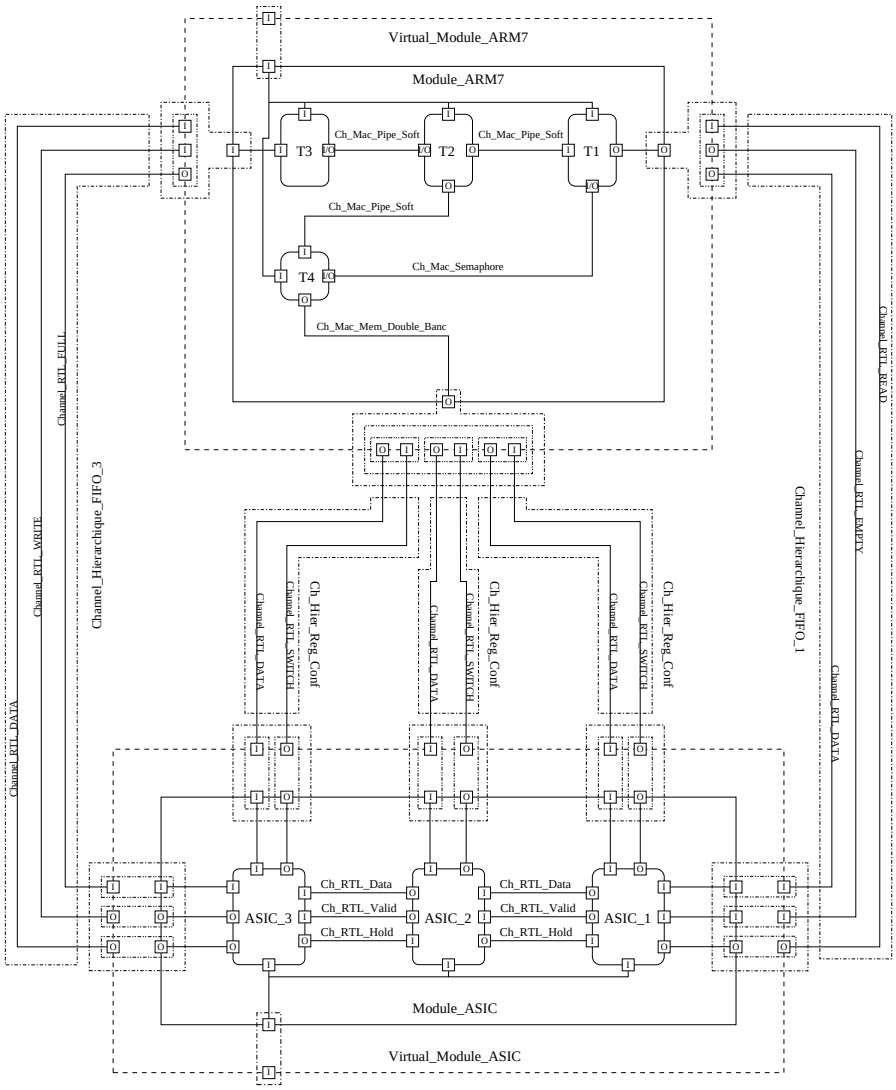


FIG. 5.6 – Première spécification du décodeur WCDMA.

Deuxième Spécification

Cette spécification (figure 5.7) diffère de la précédente par l'ajout d'un élément de communication. Dans la première spécification, l'élément de communication matérielle permettant au processeur de contrôler le pipeline n'est pas défini. Dans la seconde, cette élément de communication est défini comme un banc de registre de communication associé au « pipeline ».
Cet ajout modifie la modélisation de la communication de contrôle entre le « pipeline » et le microprocesseur. Dans le premier cas, cette communication est modélisée par des accès des éléments de calcul à leur registre de configuration. Dans le cas présent, par des accès du processeur au banc de registre de configuration.
Ces modifications d'accès entraînent le remplacement des trois canaux Ch_Hier_Reg_Conf_1, Ch_Hier_Reg_Conf_2 et Ch_Hier_Reg_Conf_3 par Ch_Hier_Shm_DB ainsi que le remplacement des ports virtuels correspondants.
Les différences entre ces deux spécifications vont entraîner des modifications dans les enveloppes de simulation et les interfaces de communication matérielles générées par les outils du flot.

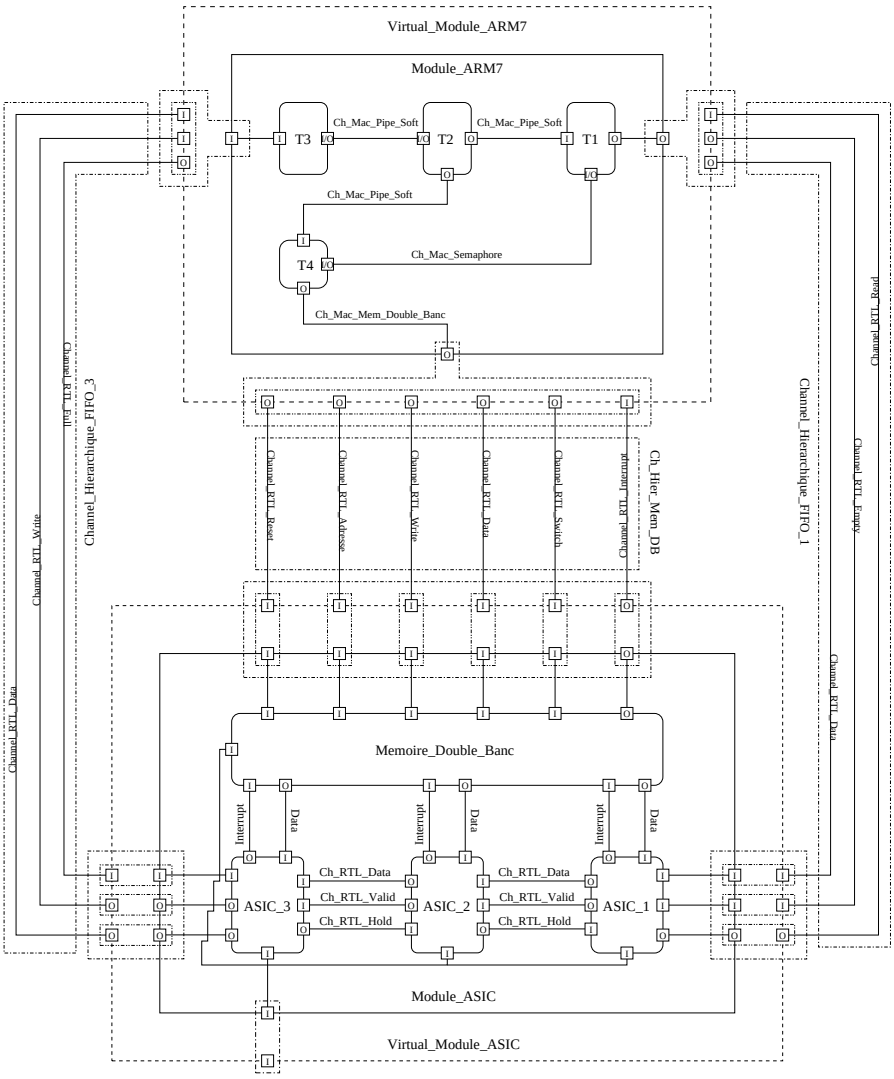


FIG. 5.7 – Deuxième spécification du décodeur WCDMA.

5.1.4 Architectures générées

Pour implémenter la première spécification, l'outil ASAG a généré la micro-architecture donnée en figure 5.8. Trois doubles bancs de registres commutables (SDBR) ont été mis en place pour répondre à chaque canal de configuration. Il en est de même pour les deux FIFO asynchrones interfaçant le processeur ARM7 au chemin de données de l'ASIC.

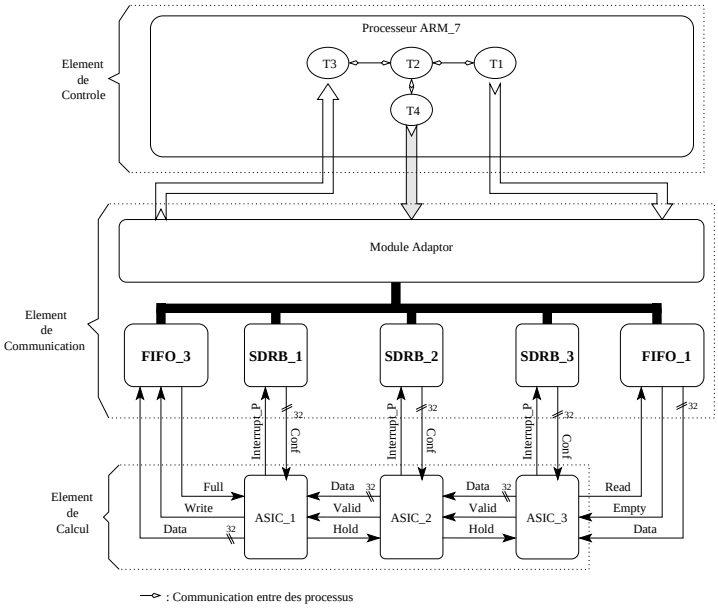


FIG. 5.8 – Première architecture du WCDMA générée.

La seconde spécification a conduit ASAG a générer la micro-architecture de la figure 5.9. Ici les doubles bancs faisant déjà partie de la modélisation de l'ASIC, les adaptateurs de canaux concernés (MMR) ne contiennent que des connexions directes de leurs ports d'entrée vers leurs ports de sortie pour les canaux en émission (du point de vue de l'ARM7) et des portes trois états pour les canaux en réception.

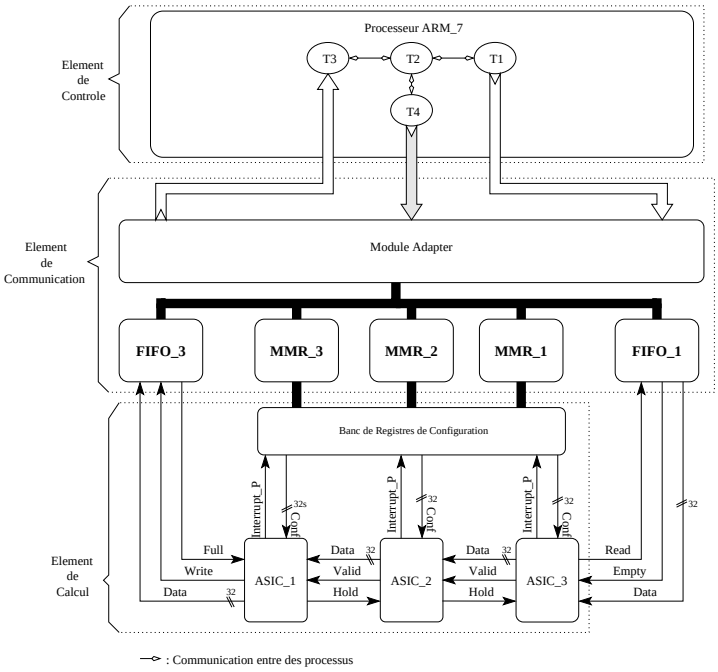


FIG. 5.9 – Seconde architecture du WCDMA générée.

5.1.5 Analyse

Le support effectif de ces deux configurations architecturales par l'outil a permis de mettre en évidence la flexibilité du flot de raffinement. Cependant, le modèle de l'application WCDMA utilisé ne reflète une réalité industrielle que par son architecture et par les protocoles utilisés. La modélisation des fonctionnalités, bien que fortement inspirée d'un modèle industriel, exhibe une complexité en deçà de la réalité. C'est pourquoi les résultats de cette étude de cas sont cantonnés à la preuve de flexibilité, tandis qu'une application plus complexe fut ultérieurement étudiée en vue de l'analyse qualitative des architectures générées : le modem VDSL.

5.2 Un modem VDSL

Cette application est basée sur la conception d'un modem VDSL développé par l'équipe AST de ST Microelectronics [60, 61] d'après le standard [66]. Les résultats concernant la modélisation et le raffinement de ce système ont été publiés dans [16, 17, 20].

5.2.1 Présentation de l'application : le VDSL

La technologie VDSL : une nouvelle technologie pour la communication haut débit sur ligne téléphonique

VDSL signifie «Very hight data rate Digital Subscriber Line». C'est une technologie qui permet la transmission de données à grande vitesse sur des lignes téléphoniques en paires torsadées, avec un débit adaptatif selon l'état de cette ligne. Elle fait partie des technologies dites xDSL, toutes dérivées de la technologie DSL utilisée pour supplanter les liaisons numériques Réseau Numérique à Intégration de Service (RNIS). Cette famille regroupe quatre technologies différentes : l'ADSL, HDSL, SDSL et VDSL. Actuellement, l'Asynchronus Digital Subscriber Line (ADSL) est la technologie la plus déployée commercialement. Le VDSL est une technologie voisine permettant des débits plus élevés. Avec l'ADSL, les débits maximums sont de 800 Kbps en émission et de 8 Mbps en réception tandis que le VDSL permet d'atteindre les vitesses maximales de 26 Mbps en émission et 52 Mbps en réception. Le principe du VDSL est de substituer à la traditionnelle modulation du courant électrique une modulation nommée DMT pour Discrete MultiTone modulation. Le DMT est une modulation quadratique d'amplitude (QAM) qui tronçonne la bande passante en tronçons de 4 KHz. Avec cette technologie, l'abonné pourra téléphoner et se connecter à internet en même temps sur une seule prise téléphonique classique. A l'arrivée de la ligne de cuivre, les fréquences vocales sont acheminées vers le réseau téléphonique tandis que les autres données sont dirigées vers le réseau internet. Sans recâblage ni changement de postes téléphoniques, l'utilisation d'un modem spécifique suffit pour utiliser cette technologie.

L'architecture VDSL

Une des particularités de ce système est de calculer le débit optimal en fonction de la qualité de la ligne (longueur, bruit etc.). L'architecture du système est présentée par la figure 5.10. . Les parties émission, réception, codage et décodage

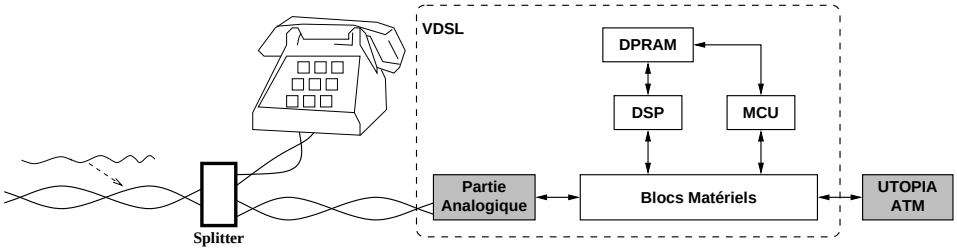


FIG. 5.10 – Architecture initiale du VDSL

sont implémentées par trois blocs matériels et un DSP avec une Dual-port Random Access Memory (DPRAM). Le microcontrôleur (MCU) permet, avec une analyse périodique, de mesurer le débit maximal supporté par la ligne. Il est aussi chargé de contrôler, configurer et synchroniser la chaîne de transmission en fonction de l'état de la ligne.

Le sous-ensemble : utilisation de deux processeurs ARM

Afin de rendre plus flexible la chaîne de traitement réaliser par l'assemblage de blocs matériels, il a été proposé d'ajouter un processeur ARM en charge de configurer dynamiquement ce chemin de données (figure 5.11).
La partie du système à concevoir était donc composée de deux processeurs et d'un pipeline de blocs matériels implémentant la chaîne de transmission.

Description du sous-ensemble de test

Dans l'objectif d'évaluer le flot de conception SLS, un sous-ensemble de l'application VDSL a été développé. Le but était de conserver l'ensemble de deux processeurs et d'un bloc communicant, d'essayer de garder les principes et difficultés d'origine pour constituer un véritable test pour le flot.

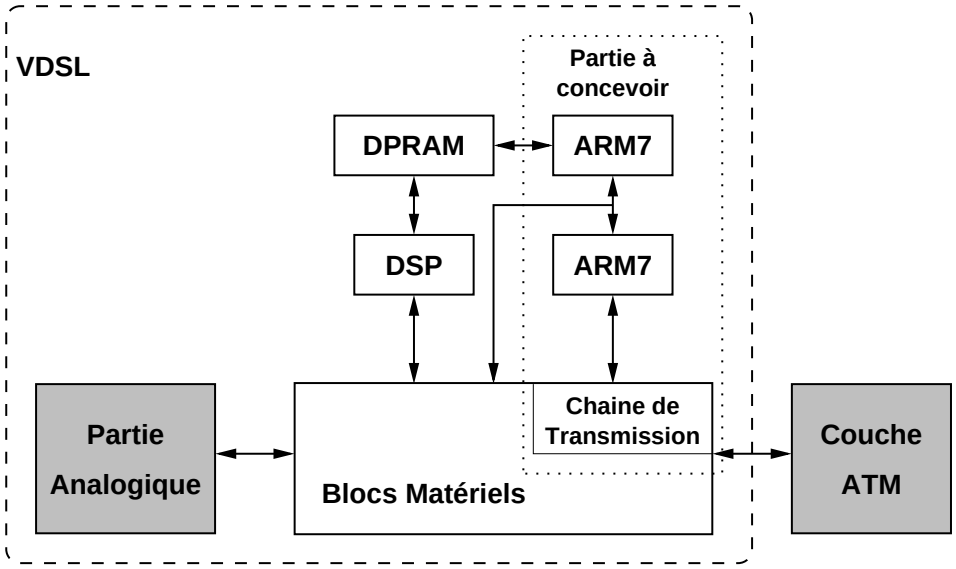


FIG. 5.11 – Architecture modifiée du VDSL.

La structure et le partitionnement logiciel/matériel du sous-ensemble de test ainsi que le comportement des tâches logicielles sont présentés ci-dessous.

Structure et partitionnement La figure 5.12 présente une description du sous-ensemble réalisé. Le système est modélisé par un assemblage de modules, nets et ports.

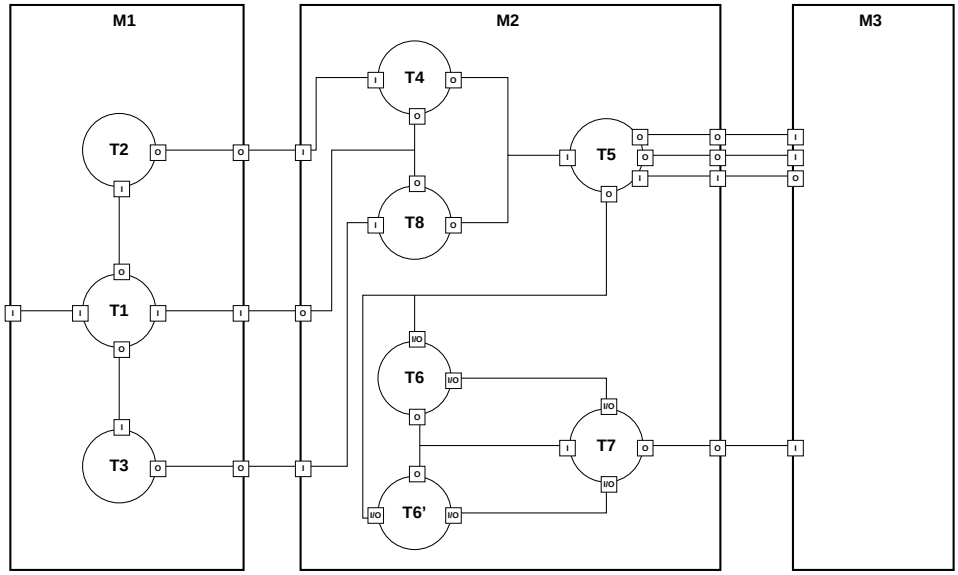


FIG. 5.12 – Description de la structure du sous-ensemble de test.

Les deux premiers modules (M1, M2) représentés par de grands rectangles correspondent aux processeurs et le troisième (M3) à la chaîne de transmission. Chaque module processeur comporte un ensemble de tâches communicantes selon des protocoles de communication spécifiés. Le deuxième processeur envoie des paramètres pour configurer et synchroniser M3. Il est aussi chargé d'envoyer périodiquement des données (Voc bytes) permettant au modem d'envoyer des messages vers son homologue distant.

Les tâches sont représentées par des cercles, leur communication est modélisée par des ports et des connexions respectivement représentés par des petits carrés dans lesquels on indique le sens de transfert des données (`I` : Input, `iO` : Output etc.) et des traits.

Description du comportement des tâches Le tableau 5.2 présente le comportement de chacune des tâches.

TAB. 5.2 – VDSL - Comportement des tâches.

	Entrées	Sorties	Actions
T1	- Un signal de l'extérieur. - Un compte rendu de T4 et de T8.	- Un ordre de génération de configuration à T2. - Un signal de Reset à T3.	- Envoie un signal de Reset à T3. -Si réception d'un signal de l'extérieur, envoi d'un signal à T2. - Si réception d'un signal de l'extérieur, envoi d'un signal à T3.
T2	- Un ordre de génération de configuration de T1.	- Un numéro de configuration à T4	- Si réception d'un signal : - Génération aléatoire d'un numéro entre 1 et 5. - Envoi du numéro à T4.
T3	- Un signal de Reset.	- Le numéro de configuration d'initialisation.	- Si réception d'un signal de Reset - Envoi du numéro de configuration d'initialisation.
T4	- Un numéro de configuration de T2.	- Un numéro de configuration à T5. - Un compte rendu de l'état à T1.	- Si réception d'un nouveau numéro de configuration : - Envoi du numéro de configuration à T5. - Envoi du numéro de configuration à T1.
T5	- Un numéro de configuration de T4 ou T8.	- Des paramètres de configuration vers les registres de M3.	Si réception d'un nouveau numéro de configuration : - Demander au DATA PATH de passer à l'état V_INIT (écrire dans le registre TCS). - Attendre le changement d'état (attendre que TXIN=V_INIT). - Attendre 10 ms. - Ecriture des paramètres dans le registre. - Attendre le changement d'état V_O/R_DATA - Ecrire DEFAULT dans le TCS. - Donner la valeur 2 dans la shm de synchronisation.
T6	- Une indication de lecture mémoire effectuée par T7.	- Les VOC bytes de synchronisation dans la mémoire partagée.	- Si T7 n'est pas en train de lire : - Si la shm de synchro vaut 1 ou 2 : - Initialisation des messages de synchronisation. - Décrémentation dans la shm de synchro de 1. - Sinon : - Ecriture des messages de sécurité dans la mémoire partagée.
T6'	- Une indication de lecture mémoire effectuée par T4.	Les VOC bytes de synchronisation dans la mémoire partagée.	Si la shm de synchro vaut 1 ou 2 :

suite sur la page suivante

suite de la page précédente			
	Entrées	Sorties	Actions
			- Initialisation des messages de sécurité. - Décrémentation dans la shm de 1. Sinon : - Écriture des messages de sécurité dans la mémoire partagée.
T7	- Les VOC bytes de sécurité et de synchronisation par la mémoire partagée.	- Une indication de lecture dans la mémoire affectée à T6 et T6'. - Les VOC bytes dans la FIFO de M3.	- Lecture de VOC bytes dans la mémoire partagée. - Indique à T6 et T6' que la lecture a été effectuée. - Écriture des VOC bytes dans la FIFO de M3.
T8	- Le numéro de configuration d'initialisation de T3.	- Le numéro de configuration d'initialisation à T5. - Un compte-rendu de l'état de T1.	Si réception d'un nouveau numéro de configuration : - Envoi de ce numéro à T5. - Envoi du numéro à T1.

5.2.2 Un sous-ensemble de test significatif

Malgré les simplifications, le sous-ensemble choisi reste suffisamment sophistiqué pour constituer un test significatif du flot.

Une spécification multiniveau Le module M3 aussi appelé Intellectual Property (IP) est un composant décrit au niveau RTL alors que les autres modules sont décrits au niveau *Message* (voir chapitre 3.2.3). Cela permet de tester la capacité du flot à réaliser un système à partir de composants de niveaux d'abstraction différents.

Diversité des protocoles de communication Un nombre important de protocoles de communication a été utilisé afin d'évaluer la difficulté d'utiliser une grande variété de types de communication.
Les tâches communiquent par l'intermédiaire de « pipes » (tubes en français), signaux, mémoires partagées simples et mémoires partagées gardées par sémaphores. La communication entre processeurs est réalisée par pipes synchronisés par « handshake » (poignée de main). Le deuxième processeur utilise, pour contrôler l'IP, l'écriture et la lecture dans des registres ou dans une FIFO.

5.2.3 Expérimentation : conception du système VDSL

Ce chapitre présente des détails sur la réalisation de l'expérience ainsi que des résultats.

La spécification La figure 5.13 représente la spécification VADeL sous forme graphique : les parties grisées correspondent aux enveloppes (wrappers) des modules virtuels.
On reconnaît les deux modules processeurs M1 et M2 ainsi que le bloc matériel avec le module M3. Les protocoles de communication sont spécifiés sur les nets :

`event` pour la transmission d'un événement permettant la synchronisation de fonctionnalités distribuées sur deux processeurs ;

`ASFIFO` pour l'échange de donnée au travers d'une file implémentée en matérielle avec un accès synchrone (coté producteur de données) et asynchrone par protocole rendez-vous 4 phases (coté consommateur) ;

`MMR` pour la scrutation de registres d'états et l'utilisation de registres de configuration ;

`GBuffer` spécifie l'usage d'un buffer FIFO dont le remplissage est exclusif jusqu'à l'occupation totale (FIFO pleine), puis la consultation est autorisée et exclusive jusqu'à vider la FIFO.

On remarque aussi, dans l'interface de l'enveloppe de `M2`, un SAP dont l'annotation `WD` (pour « Watchdog ») spécifie un service de chronométrage que le décompte d'un temporisateur implémentera.

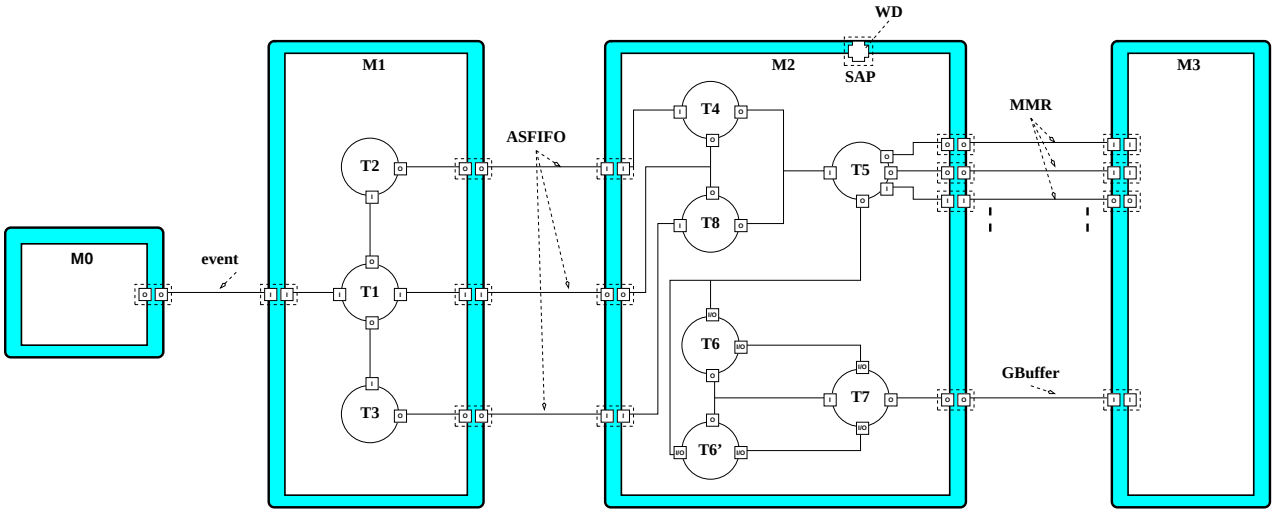


FIG. 5.13 – Spécification de l'application en VADeL.

Module	Nombre de portes logiques équivalentes	Surface (μm^2)	Période d'horloge minimale (ns)
M1	3284	8168	5,95
M2	3795	5100	6.16

TAB. 5.3 – Résultats de synthèses des CC.

La génération d'interfaces matérielles

Résultats La figure 5.14 présente la micro-architecture obtenue après génération des interfaces matérielles.
Les éléments composant les interfaces ont été générés en code VHDL puis synthétisés. Les assemblages d'interfaces sont aussi complètement décrits en VHDL.
La génération de ces interfaces a nécessité 5 minutes (interactions avec l'utilisateur comprises). La synthèse RTL de ces interfaces peut être résumée par les quelques chiffres présentés dans le tableau 5.3. En terme de chemin critique, la fréquence d'horloge de 140 Mhz supportés par les deux CC est supérieure aux 66 Mhz d'un processeur ARM7TDMI (fabriqué avec la même technologie). Bien que ne représentant que 5% de la surface totale, ces CC peuvent dans certains cas paraître d'un surcoût inacceptable. Une analyse plus détaillée de cette occupation surfacique révèle que 87.9% de cette surface est utilisée pour la réalisation de quatre files (FIFO) dont la taille cumulée et spécifiée par le concepteur, s'élève à 3200 bits. Aussi est-il nécessaire de rappeler que dans cette configuration de l'architecture, ce sont les CA qui ont la responsabilité du contrôle des protocoles. Il est bien entendu possible de générer des implémentations moins coûteuses à base de registre pilotant les signaux des protocoles, comme le fait COWare N2C, mais ce serait au détriment des performances du logiciel.
Afin d'aider à quantifier le gain obtenu par l'embarquement du contrôle des protocoles dans les CA par rapport a des implémentations logicielles de ce même contrôle (telles que les offrent N2C), peut être mesuré grâce à la loi d'Amdahl (équation 5.1)[36]. Pour cela, les implémentations logicielles et matérielles du contrôle des protocoles « poignée de main » 4 phases maître-émetteur et esclave-récepteur vont être comparées. Les délais d'occupation du processeur inhérents aux deux implémentations de ces protocoles sont donnés dans les tableaux 5.4 et 5.5. La signification des colonnes de ces tableaux est la suivante :

– Les colonnes *OP.* indiquent l'état courant du protocole. Ainsi */Ack ?* signifie l'attente de l'infirmation du signal d'acquittement, *Req* représente l'affirmation du signal de requête et *Data* modélise les accès aux données en lecture ou en écriture.
– Les colonnes *nb* donnent le nombre d'itération dans l'état courant. Les valeurs n et m sont relatives à la synchronisation des terminaisons communicantes et donc dépendantes de l'application.
– Pour les colonnes D_u, elles présentent le coût unitaire de l'état courant.
– Les dernières colonnes : $\sum D$ contiennent le cumul des délais.

Les différents délais introduits sont :

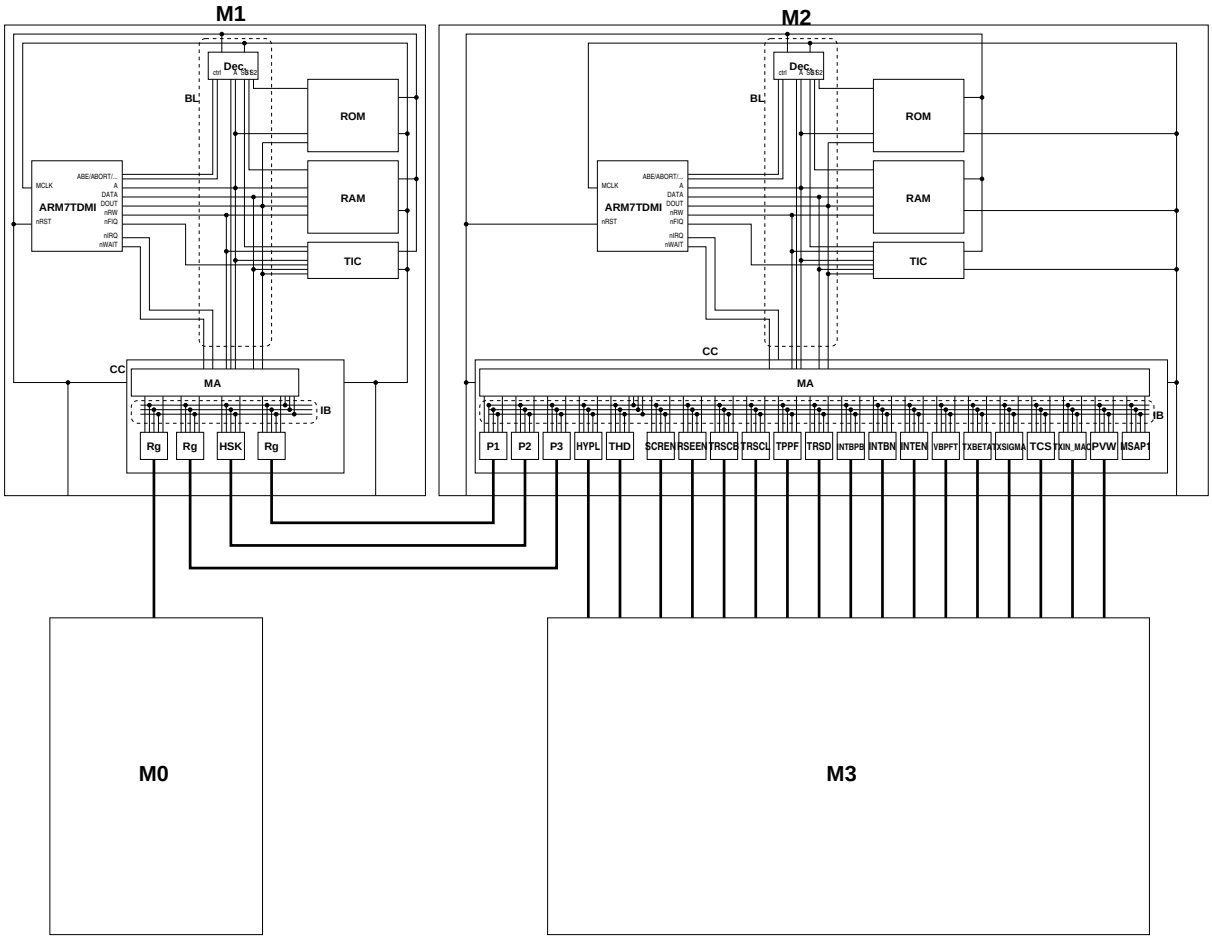


FIG. 5.14 – La micro-architecture du VDSL générée avec des communications p-à-p.

$$\begin{aligned}\text{Accélération} &= \frac{\text{Performance pour la tâche entière en utilisant la caractéristique quand c'est possible}}{\text{Performance pour la tâche entière sans utiliser la caractéristique}} \\ &= \frac{\text{Temps d'exécution pour la tâche entière sans utiliser la caractéristique}}{\text{Temps d'exécution pour la tâche entière en utilisant la caractéristique quand c'est possible}}\end{aligned} \tag{5.1}$$

Contrôle logiciel				Contrôle matériel			
Op.	nb	D_u	$\sum D$	$\sum D$	D_u	nb	Op.
/Ack ?	n	$D_r + D_b$	$n * D_r + (n-1) * D_b$	D_w	D_w	1	Data
Data	1	D_w	$n*D_r+(n-1)*D_b+D_w$	$D_w + D_{cs}$	D_{cs}	1	Gel tâche
Req	1	D_w	$n * D_r + (n-1) * D_b + 2 * D_w$	$D_w + D_{cs} + D_{it}$	D_it	1	Traitement IT
Ack ?	m	$D_r + D_b$	$(n+m) * D_r + (n+m-2) * D_b + 2 * D_w$	$D_w + D_{it} + 2 * D_{cs}$	D_{cs}	1	Réveil tâche
/Req	1	D_w	$(n+m) * D_r + (n+m-2) * D_b + 3 * D_w$				

TAB. 5.4 – Délais des implémentations HW/SW du Poignée-de-main/Maître/Émetteur.

D_r : délai propre à un accès en lecture ;

D_w : délai d'un accès en écriture ;

D_b : délai d'un branchement conditionnel ;

D_{cs} : délai d'un changement de contexte ;

D_{it} : délai nécessaire au processeur pour reconnaître et traiter une requête d'interruption.

Les accélérations peuvent maintenant être calculées en se basant sur les valeurs numériques exprimées en nombre de cycles horloge, dans le tableau 5.6.
Ces valeurs proviennent des traces de co-simulation *Micro-Architecture* et correspondent aux valeurs données dans [3] pour les délais propres au processeur ARM7 et dans [29] pour le changement de contexte.
Avec ces chiffres les expressions des délais d'occupation du processeur sont réduites à l'équation 5.2 pour l'implémentation matérielle du maître-émetteur, l'équation 5.3 pour son implémentation logicielle. De même, pour le protocole esclave-récepteur les formules de ces délais se réduisent à l'équation 5.4 pour son implémentation matérielle et 5.5 pour l'implémentation logicielle.
L'accélération A est donnée par l'équation 5.6. Cette accélération devient intéressante ($A > 1$) pour des latences d'attentes (en cycle horloge) de l'interlocuteur n et m telles que $m + n > 32$ cycles.
On pourra donc abuser des interruptions lorsque la fréquence de ces échanges est faible et les données volumineuses.

Alternative Dans cette application, une partie des communications est consacrée à l'échange de données de configuration ou de contrôle. De tels échanges ne requièrent pas une grande bande passante, aussi leurs

Contrôle logiciel				Contrôle matériel			
Op.	nb	D_u	$\sum D$	$\sum D$	D_u	nb	Op.
Req ?	n	$D_r + D_b$	$n * D_r + (n-1) * D_b$	D_{cs}	D_{cs}	1	Gel tâche
Data	1	D_r	$(n+1)*D_r+(n-1)*D_b$	$D_{it} + D_{cs}$	D_{it}	1	Traitement IT
Ack	1	D_w	$(n+1) * D_r + (n-1) * D_b + D_w$	$D_{it} + 2 * D_{cs}$	D_{cs}	1	Réveil tâche
/Req ?	m	$D_r + D_b$	$(n+m+1) * D_r + (n+m-2) * D_b + D_w$	$D_{it} + 2 * D_{cs} + D_r$	D_r	1	Data
/Ack	1	D_w	$(n+m+1) * D_r + (n+m-2) * D_b + 2 * D_w$				

TAB. 5.5 – Délais des implémentations HW/SW du Poignée-de-main/Esclave/Réception.

Lecture	D_r	2
Écriture	D_w	1
Test	D_b	5
Changement de contexte	D_{cs}	59+36
Traitement d'interruption	D_{it}	27

TAB. 5.6 – Durées (en cycles horloge) d'exécution des opérations sur un processeur ARM7TDMI.

$$D_{MO_{HW}} = 218 \tag{5.2}$$

$$D_{MO_{SW}} = (n+m) * 7 - 7 \tag{5.3}$$

$$D_{SI_{HW}} = 219 \tag{5.4}$$

$$D_{SI_{SW}} = (n+m) * 7 - 6 \tag{5.5}$$

$$A = \frac{(n+m) * 7 - 6}{218} \tag{5.6}$$

implémentations peuvent être concentrées sur une unique ressource physique : un bus partagé.
Une mise en œuvre de cette application utilisant un bus AMBA High-Bandwidth (AHB) pour le contrôle et une connexion point à point (pour les *VOC bytes* émis par **T8** vers **M3**) est en cours d'étude. L'architecture alors ciblée est illustrée par la figure 5.15.

5.3 Évaluation et perspectives

L'expérimentation du flot avec l'application VDSL a été l'occasion de faire une première analyse qualitative du flot et de prévoir les extensions possibles. Malgré une mise au point encore perfectible du flot, nous pouvons déjà évaluer les avantages et les inconvénients qu'apporte cette méthodologie.

5.3.1 Les avantages

L'analyse de cette étude de cas a permis la mise en évidence de points positifs relatifs aux modèles de représentations, à l'organisation des bibliothèques, à ASAG et à l'accélération de cette étape de conception.

Les modèles de représentations

Pour la conception d'un même système, il est possible de changer facilement de composants, de protocoles de communication et de paramètres des ressources (taille de FIFO etc.) par simple modification des annotations du modèle de niveau *Message*. Il est possible de changer de langage de spécification mais aussi de langages de développement des éléments de bibliothèque, grâce au découplage comportement/communication apporté par COLIF et au support de tout langage cible par les outils de macro-génération (*RIVE*, *M4*).

Les bibliothèques

La bibliothèque est extensible. Il est en effet toujours possible d'ajouter de nouveaux éléments. Si le concepteur décide d'adopter un protocole de communication non supporté, il lui suffit de concevoir les éléments permettant son implémentation, puis de les intégrer dans les bibliothèques. Cela permet de faire évoluer les outils dans le temps, mais surtout de laisser la possibilité de développer des applications très spécifiques en continuant à bénéficier des avantages de la génération automatique d'interfaces.
Le système d'annotation de paramètres permet une flexibilité pour l'optimisation d'un système. Il est ainsi possible de changer facilement, par exemple, la taille d'un bus, la profondeur d'un buffer, la priorité d'un canal par de simples changements de paramètres lors de la spécification.

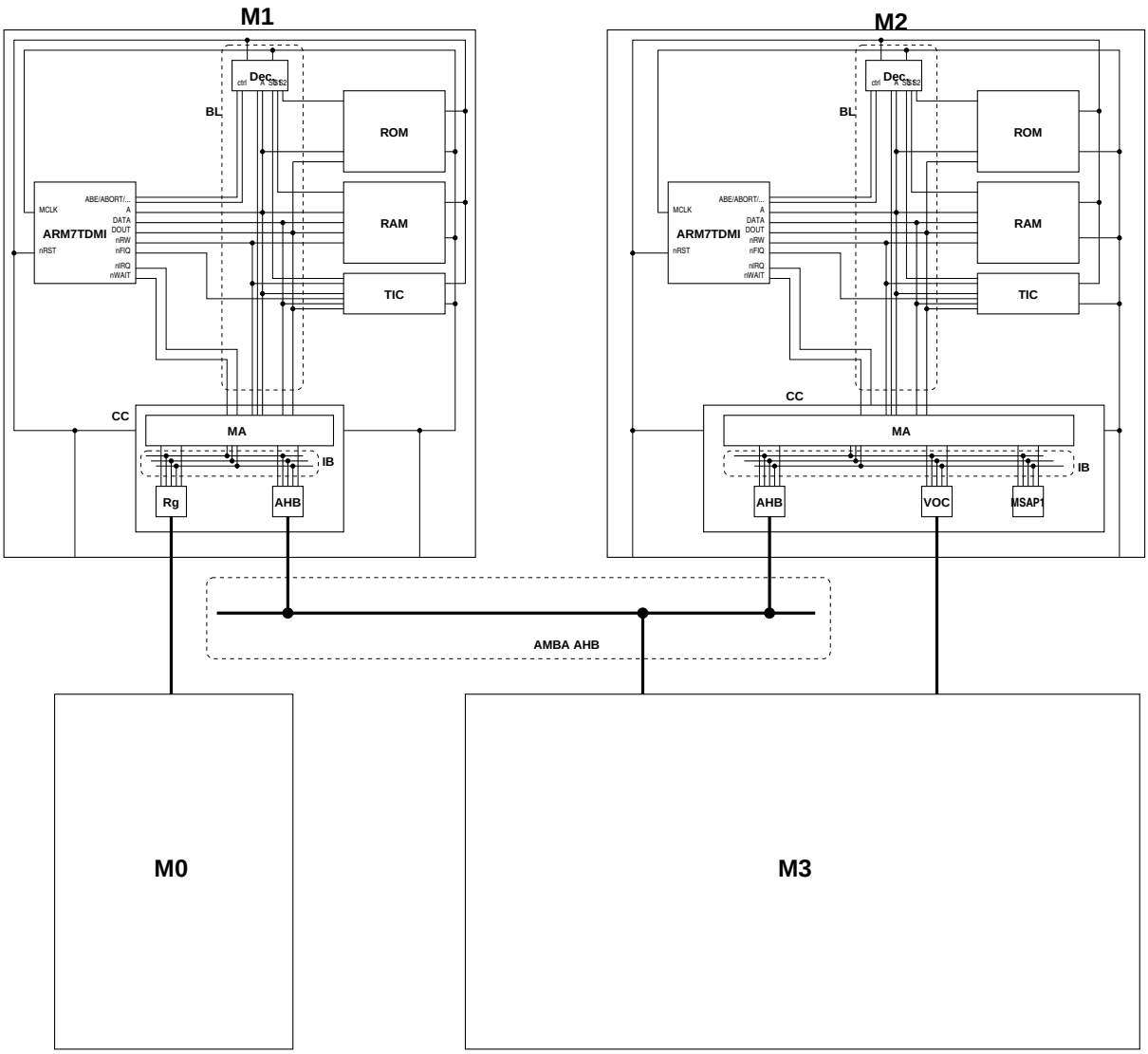


FIG. 5.15 – La micro-architecture du VDSL générée avec des communications AMBA.

ASAG

La flexibilité d'ASAG est prouvée par cette étude de cas. Il est alors intéressant de comprendre comment cette flexibilité est obtenue en identifiant les parties fixes du système ainsi que les parties modifiables. Les parties fixes sont principalement le principe de génération d'interfaces de communication pour lier les composants et les divers algorithmes d'assemblage.
La génération d'interfaces est un concept clé pour la flexibilité, on ne doit pas avoir besoin de modifier un IP lors du changement du processeur avec lequel il communiquait. La génération automatique d'interfaces permet de changer aisément de composant et d'accélérer les itérations lors de l'exploration d'architecture.
L'assemblage des interfaces permet, à partir d'un même ensemble d'éléments, de composer un nombre important d'architectures. Par exemple, pour la génération de la partie matérielle d'une interface, si au lieu d'utiliser des interconnexions point-à-point simples, on décide d'utiliser un bus partagé, il suffit d'ajouter un ensemble d'éléments piochés dans la bibliothèque permettant la connexion spécifique à ce type de bus. C'est comme le jeu de lego, un concepteur expérimenté peut à partir d'un même ensemble de pièces construire un système complexe.

Réduction du temps de conception

L'étape d'intégration (assemblage de composants existants) du modem VDSL, dans sa première version (un MCU, un DSP et le pipeline) a nécessité une semaine à un concepteur industriel confirmé.
Avec le flot de conception SLS, l'intégration de l'évolution (processeur supplémentaire) et restriction (seul la chaîne émettrice est traîtée) de cette application a été réalisée en 40 heures, incluant la conception des blocs d'adaptation (5 CA).
L'ajout de ces blocs sous une forme paramètrable et spécialisable a demandé une quinzaine de jours. La modélisation en de l'application nécessita une demi-journée à La mise en place du modèle d'entrée a réclamé une demi-journée à un concepteur aguerri au langage VADeL et aux annotations de raffinement (tailles FIFOs, adresses de données etc.). La génération des enveloppes matérielles prend 5 minutes avec un PC à intel Pentium III 500 MHz, 128Mo de mémoire et tournant sous Linux.
Cette méthodologie permet une réduction significative du temps de conception des systèmes multiprocesseurs monopuce, puisque la conception de l'application a demandé $\frac{35}{12+\frac{5}{60}} = 2.9$ fois moins d'efforts qu'avec une méthode classique lors de sa première intégration (en considérant la bibliothèque suffisante) et $\frac{35}{\frac{5}{60}} = 60$ fois moins de temps pour toutes les itérations suivantes.

5.3.2 Limitations

L'automatisation de la conception d'un système en utilisant des bibliothèques pose le problème de choix d'implémentations. D'une part, il faut développer des bibliothèques conséquentes pour offrir un choix d'implémentation de protocoles suffisamment riche au concepteur. D'autre part si un protocole de communication possède plusieurs implémentations, il faut savoir jusqu'où va l'automatisation, où s'arrête la liberté du concepteur. Pour l'instant chaque protocole ne possède qu'une implémentation (c-à-d. un seul CA compatible avec un bus interne et avec un pilote de communication).
Un autre problème est lié à la difficulté de validation des éléments de bibliothèque. En effet, même si une automatisation à base d'assemblage de composants issus de bibliothèques peut hériter d'une bonne fiabilité par la réutilisation de composants bien connus et définis, ces derniers doivent être réputés fortement validés. De même l'assemblage de ces composants n'est pas, jusqu'à preuve du contraire, à l'abri de malfaçons. Ces problèmes de validation sont ici d'autant plus cruciaux que les composants de bibliothèques sont configurables.
La méthodologie actuelle ne propose pas de solutions pour le choix du partitionnement logiciel matériel. Il n'est pas non plus possible d'obtenir une allocation dynamique des tâches sur les différents processeurs ni de concevoir un système d'exploitation gérant l'ensemble des processeurs.

5.4 Perspectives

Les perspectives présentées concernent uniquement la génération des adaptateurs matériels de communication.

5.4.1 Améliorations futures de l'outil et de la bibliothèque

Les applications ont permis de valider la méthodologie et d'éprouver l'outil et la bibliothèque. Mais elles ont surtout mis en évidence quelques faiblesses et lacunes :

- L'algorithme de mise à l'échelle est basé sur un attribut dont la valeur est statique, ce qui engendre des architectures encore trop régulières. L'utilisation de scripts (tels que du code *Python* utlisant des API) en lieu et place de cette valeur permettrait une flexibilité architecturale illimitée.
- L'interface entre l'outil de génération et les macro-modèles est un jeu statiquement défini de paramètres envoyés depuis ASAG vers l'outil externe d'expansion. Cet échange est trop contraignant, et sera remplacé par un mécanisme client (outil d'expansion) - serveur (ASAG) de paramètres. Les communications entre ces deux outils pourront alors être interactives autorisant ainsi des configurations plus détaillées des modèles de composants à générer.
- Le support de protocoles de standards d'interfaces (VCI, OpenCore), par l'ajout de modèles de CA en bibliothèque.
- Des implémentations orientées consommation, avec l'utilisation de CA capable de générer des signaux d'horloges dont la fréquence évolue en fonction du taux d'occupation de la FIFO interne au CA.
- Des implémentations fortement orientées performances des CC avec l'utilisation d'un bus micro-commuté faiblement parallèle et l'utilisation de DMA concurrents enfouis.
- La validation des architectures générées n'est actuellement possible qu'à l'aide de cosimulation *Micro-Architecture* (pure ou accélérée par une simulation fonctionnelle du logiciel). Cette approche est trop lente pour permettre l'utilisation d'un jeu de tests ayant une couverture acceptable, en des temps raisonnables.
  Deux solutions sont envisagées :
  1. l'émulation (ou coémulation lorsque certaines fonctionnalités du système sont simulées sur une station) d'architecture par son implémentation sur des cartes de prototypage ;
  2. la validation formelle de l'architecture matérielle par :
     - la validation des composants matériels aidée par la génération automatique de modèles de propriétés formelles en parallèle des modèles d'implémentation. La conformité de ces modèles peut alors être automatiquement par des environnements de preuves formelles.
     - la vérification de la cohérence des interfaces logicielles $\leftrightarrow$ matérielles pourra quant à elle être réalisée par un outil de test de règles (générées en parallèle de l'assemblage des modèles d'implémentation) en cours de développement dans l'équipe SLS.
- les composants générés sont numériques et synchrones, mais l'extension des domaines d'applications des bibliothèques par l'ajout de nouveau macro-modèles pourra permettre la mise en œuvre de composants :
  - numériques asynchrones (pouvant être modélisés par des langages tels que le CSP) ;
  - analogiques, micro-mécaniques, ou micro-optiques.

5.4.2 Évaluation des performances, consommation et coûts des implémentations de protocoles de communication pour instrumenter l'exploration d'architectures

Pour permettre au concepteur de guider ses choix dans son exploration d'architecture, il faut pouvoir évaluer les performances, la consommation et le coût des architectures générées en des temps réduits. Il faudrait développer une méthode d'évaluation systématique des implémentations logicielles/matérielles des protocoles. Il serait alors possible de modifier les outils afin d'obtenir une sélection automatique d'éléments de bibliothèque en fonction du type de protocole et d'une spécification de contraintes.

5.5 Conclusion sur les études de cas

Nous avons, au cours de ce chapitre, confronté notre modèle de représentation d'architectures et notre flot de raffinement à la conception de 1^o un modem WCDMA et de 2^o un modem VDSL. Le support effectif de l'hétérogénéité des modules et de leurs communications est constaté. L'obtention de modèles RTL en quelques minutes suffit à prouver l'accélération de cette étape d'intégration des composants. Le VDSL, avec ses besoins propres en services (protocoles) de communication, a permit de vérifier l'ouverture des bibliothèques grâce à l'ajout de 6 CA en moins de 2 semaines. Cette flexibilité n'altère en rien la qualité des réalisations comme nous la démontrer les résultats de synthèse et de simulation, ainsi que la comparaison avec une approche plus monolithique telle que N2C.

Cependant il serait faux et utopique de prétendre à une réponse exhaustive au problème d'intégration. Les limitations de notre approche sont relatives à 1^o la disponibilité des composants de bibliothèque et 2^o à leur validation.
Alors qu'un acheminement vers la complétude de la bibliothèque repose sur un simple mais fastidieux enrichissement manuel, le développement d'une méthode pour sa validation nécessite une réflexion à plus long terme.

Chapitre 6

Conclusions

Sommaire

« Le sage affirme avec raison que « L'expérience est une lanterne que l'on s'accroche dans le dos et qui n'éclaire guère que le chemin déjà parcouru. »... Au moins le lampiste a-t-il en l'occurence la satisfaction de baliser la route de ceux qui le suivent ! »

– Philippe KARL

6.1 Les systèmes embarqués spécifiques : maîtrise de la complexité

Les systèmes embarqués sont les parties électroniques qui prennent place progressivement et naturellement dans les objets usuels allant des téléphones aux automobiles.
Récemment la demande pour les systèmes embarqués et le nombre de fonctionnalités souhaitées s'est fortement accrue tandis que les délais de conception requis diminuent. Des architectures multiprocesseurs hétérogènes semblent devenir la clé pour que les systèmes embarqués puissent supporter cette complexité. En parallèle, l'intégration a fait de grands progrès : il est maintenant possible d'intégrer sur une même puce plus de 100 millions de transistors. Il est dès lors possible d'intégrer complètement un système sur une seule puce. Cependant, les concepteurs n'arrivent plus à concevoir de tels circuits dans des délais raisonnables : ils manquent de méthodologies et d'outils (voir le premier chapitre).

6.2 Objectif du projet de l'équipe SLS

L'objectif est de fournir les méthodologies et les outils qui faciliteront et accéléreront la conception des systèmes monopuces. Pour ce faire, un flot de conception descendant est proposé. Ce flot part d'une spécification de haut niveau du système à générer, et grâce à des outils automatiques de raffinement, se propose de générer le code de bas niveau correspondant. Ce flot offre aussi la possibilité de simuler le système au cours des diverses étapes de raffinement. Il met l'accent sur les architectures multiprocesseurs, ainsi que sur les communications.
Quoique encore incomplet, ce flot présenté au premier chapitre propose déjà quelques outils automatiques de raffinement, dont un outil de génération d'architecture avec synthèse des adaptateurs matériels de communication, qui est le sujet de ce mémoire.

6.3 Revue des travaux présentés

6.3.1 Outil de génération d'architecture spécifique par l'assemblage systématique de composant

L'outil présenté au quatrième chapitre se propose de générer des architectures spécifiques à une application. Il prend en entrée une spécification abstraite annotée de l'application, et produit en sortie des modèles d'implémentation de l'architecture et des adaptateurs de communication en assemblant et en configurant des composants contenus dans une bibliothèque.

6.3.2 Bibliothèque de composants pour l'assemblage

Cette bibliothèque est présentée dans la section 4.4. Elle est le coeur de la méthodologie de génération d'architecture. Elle est constituée de deux parties : une partie donnant la description des divers éléments, et une partie contenant leur code.
La première partie contient de nombreux objets dont les principaux sont les modèles structurels de :
– architectures locales spécifiques à un processeur ;
– adaptateurs de modules capables d'interfacer un processeur et de gérer le partage de ressources ;
– adaptateurs de canaux, véritables accélérateurs matériels de communication prenant en charge le contrôle des protocoles ;
– canaux de communication pouvant contenir des sous-composants matériels (décodeurs, arbitres, routeur, concentrateur ...).

Ces éléments sont fournis avec des attributs indiquant leur compatibilité respective.
La deuxième partie contient le code correspondant aux implémentations sous forme d'entrelacement de portions de code final et de macros. Celles-ci, écrites dans un langage de macro complet, génèrent les portions à configurer de code final des modèles de composant, et permettent son adaptation à l'application.

6.3.3 Application

L'outil et la bibliothèque ont été utilisés pour deux applications : 1^o un modem VDSL et 2^o une station mobile WCDMA. Ces applications sont présentées dans le chapitre précédent. L'objectif était de générer deux adaptateurs de

communication pour deux processeurs ARM7, avec plusieurs protocoles de communication et de synchronisation. Les coprocesseurs de communication ont été rapidement générés. Ils sont de taille négligeable aux vues des autres composants de l'architecture (processeurs et accélérateurs), de plus leurs performances se comparent équitablement à celles obtenues avec des méthodes.

6.4 Perspectives

L'idée d'utiliser des architectures multiprocesseurs pour augmenter la puissance de calcul n'est pas nouvelle : elle a fait l'objet de nombreuses recherches depuis les débuts de l'informatique. Elle n'a cependant jamais connu un grand succès, principalement pour trois raisons :

– La communication entre les processeurs est souvent un goulet d'étranglement.
– La programmation de telles architectures est difficile.
– Les progrès en intégrations ont jusqu'à présent fait augmenter la puissance des processeurs exponentiellement, ce qui rend peu intéressant les efforts à fournir pour développer du logiciel pour une architecture multiprocesseur qui serait vite dépassée.

Il peut alors être légitime de se demander pourquoi l'approche multiprocesseur aurait des chances de réussir pour les systèmes embarqués spécifiques alors qu'elle a échoué auparavant. Nous pouvons remarquer que la difficulté pour programmer les architectures multiprocesseurs venait souvent du fait que les concepteurs voulaient exécuter un algorithme (souvent séquentiel) sur l'architecture parallèle. Pour les systèmes embarqués spécifiques l'approche est différente : le but n'est plus de pouvoir résoudre n'importe quel problème informatique, mais plutôt de fournir un certain nombre de fonctionnalités spécifiques faiblement corrélées les unes aux autres. Dans de telles architectures, chaque processeur est dédié à quelques tâches spécifiques, et peut donc être conçu ou programmé individuellement tant qu'il ne communique pas avec les autres. La conception multiprocesseur devient alors de la conception monoprocesseur répétée autant de fois qu'il y a de processeurs.
Il reste cependant la problématique des communications entre les divers processeurs : elle représente à la fois un goulet d'étranglement et une difficulté de conception ou programmation du fait de l'hétérogénéité des architectures embarquées. L'aspect hétérogène des systèmes embarqués apporte une solution simple aux problèmes de coûts : chaque partie étant spécifique à certaines fonctions, une conception intelligente de l'architecture peut favoriser des implémentations locales efficaces. L'arrivée des systèmes monopuces réduit encore davantage ces coûts.
Toutes ces perspectives sont renforcées par l'arrivée éminente d'une catégorie de systèmes surclassant en complexité tout ce qui ce fait actuellement : les **réseaux sur puce** ou « Network-on-Chip (NoC) ». Dans de tels systèmes le nombre de nœuds de calcul est tel ([7] en compte déjà 430) que seule une approche réseau avec ses modèles de calcul et de communication unifiés (ie. OSI) en apportera la maîtrise. La disponibilité d'outils et méthodes en facilitant l'implémentation sera alors grandement appréciée.

Bibliographie

Bibliographie

[1] ARM Ltd. *AMBA Rev. 2.0 Specification*. available at : http ://www.arm.com/arm/AMBA.

[2] ARM Ltd. *ARM7 Data Sheet*. available at http ://www.arm.com/ Documentation/UserMans/PDF/ARM7vC.pdf.

[3] ARM Ltd. *ARM7TDMI Data Sheet*. available at http ://www.arm.com/ Documentation/UserMans/PDF/ARM7TDMIvE.pdf.

[4] A. Baganne, J. Philippe, and E. Martin. A formal technique for hardware interface design. In *Proc. Int'l Symposium on Circuits and Systems*, June 1997.

[5] A. Baghdadi. *Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques*. PhD thesis, Institut National Polytechnique de Grenoble, May 2002.

[6] A. Baghdadi, D. Lyonnard, N. Zergainoh, and A. Jerraya. An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC. In *Proc. Design Automation and Test in Europe*, pages 55–62, Mar. 2001.

[7] R. Baines and D. Pulley. The picoArray and Reconfigurable Baseband Processing for Wireless Basestations. available at http ://www.picochip.com/content/news/IEEE_ComSoc_Jan2003.pdf, 2003.

[8] F. Belina, D. Hogrefe, and A. Sarma. *SDL with APPLICATIONS from PROTOCOL SPECIFICATION*. Carl Hanser Verlag and Prentice Hall International (UK) Ltd., 1991.

[9] T. Benner, R. Ernst, I. Konenkamp, U. Holtmann, P. Schuler, H. C. Schaub, and N. Serafimov. FPGA based prototyping for verification and evaluation in hardware-software cosynthesis. In *International Workshop on Field-Programmable Logic and Applications*, pages 251–258, 94.

[10] R. A. Bergamaschi. Automating the Drudgery in System-on-Chip Design. In *Workshop on Synthesis and System Integration of Mixed Technology (SASIMI)*, 2000.

[11] R. A. Bergamaschi and W. R. Lee. Designing Systems-on-Chip Using Cores. In *Proc. Design Automation Conf.*, June 2000.

[12] R. Berrendorf, H. C. Burg, U. Detert, R. Esser, M. Gerndt, and R. Knecht. Intel paragon XP/S - architecture, software environment, and performance. Technical Report KFA-ZAM-IB-9409, 1994.

[13] Cadence inc. *Virtuoso-XL*. available at http ://www.cadence.com/products/vlayout.html.

[14] Cadence inc. *VoltageStorm*. available at http ://www.cadence.com/products/voltagestorm.html.

[15] Cadence inc. *Virtual Component Codesign*, 2001. available at http ://www.cadence.com.

[16] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. Component-Based Design Approach for Multicore SoCs. In *Proc. Design Automation Conf.*, June 2002.

[17] W. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava. Multiprocessor SoC Platforms : A Component-Based Design Approach. *IEEE Design & Test of Computers*, 19(6), nov–dec 2002.

[18] W. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, and A. A. Jerraya. Colif : A Design Representation for Application-Specific Multiprocessor SOCs. *IEEE Design & Test of Computers*, 18(5) :8–20, sep–oct 2001.

[19] W. O. Cesario, L. Gauthier, D. Lyonnard, G. Nicolescu, and A. A. Jerraya. An XML-based Meta-model for the Design of Multiprocessor Embedded Systems. In *VHDL International User's Forum (VIUF)*, Oct. 2000.

[20] W. O. Cesario, Y. Paviot, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. HW/SW Interfaces Design of a VDSL Modem using Automatic Refinement of a Virtual Architecture Specification into a Multiprocessor SoC : a Case Study. In *Proc. Design Automation and Test in Europe*, Mar. 2002.

[21] P. H. Chou, R. B. Ortega, and G. Borrielo. Synthesis of the Hardware/Software Interface in Micro-controller based Systems. In *a determiner*, 1992.

[22] P. H. Chou, R. B. Ortega, and G. Borrielo. Interface co-synthesis techniques for embedded systems. In *a determiner*, 1995.

[23] CORBA Services. Corba services ; common object services specification. Technical report, Object Management Group, 1997.

[24] W. O. Césario. *Colif Semantic Refernce Manual*. TIMA lab., June 2000.

[25] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann Publisher.

[26] S. Dutta, R. Jensen, and A. Rieckmann. Viper : A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems. *IEEE Design & Test of Computers*, 18(5) :21–31, sep–oct 2001.

[27] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, 10(3) :64–75, Dec. 1993.

[28] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, (21) :948–960, Sept. 1972.

[29] L. Gauthier. *Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systêmes embarqués spécifiques*. PhD thesis, Institut National Polytechnique de Grenoble, Dec. 2001.

[30] F. Gharsali, D. Lyonnard, F. Rousseau, and A. Jerraya. Unifying Memory and Processor Wrapper Architectyre in Multiprocessor SoC Design. In *Proc. Int'l Symposium on System Synthesis*, Oct. 2002.

[31] F. Gharsali, S. Meftali, F. Rousseau, and A. A. Jerraya. Embedded Memory Wrapper Generation for Multi-processor SoC Design. In *Proc. Design Automation Conf.*, June 2002.

[32] T. D. Givargis, F. Vahid, and J. Henkel. Instruction-based System-level Power Evaluation of System-on-a-chip Peripheral Cores. In *Proc. Int'l Symposium on System Synthesis*, 2000.

[33] P. Guerrier and A. Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *Proc. Design Automation and Test in Europe*, 2000.

[34] D. Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, (8) :231–274, 1987.

[35] J. Henkel, T. Benner, R. Ernst, W. Ye, N. Serafimov, and G. Glawe. COSYMA : a software-oriented approach to hardware/software codesign. *Journal of Computer and Software Engineering*, 2(3) :293–314, 1994.

[36] J. L. Hennessy and D. Patterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 2 edition, 1996.

[37] F. Hessel, P. Coste, G. Nicolescu, P. Le Marrec, N.-E. Zergainoh, and A. A. Jerraya. Multi-level Communication Synthesis of Heterogeneous Multilanguage Specifications. In *Proc. Int'l Conference on Computer Design*, Sept. 2000.

[38] D. Hommais. *Réalisation et Evaluation des communications dans les systèmes mixtes Matériel-Logiciel*. PhD thesis, Paris VI, 2001.

[39] D. Hommais, F. Pétrot, and I. Augé. A tool box for system level communication synthesis. In *Proc. IEEE International Workshop on Rapid System Prototyping*, 2001.

[40] K. Hwang. *Advanced Computer Architecture : Parallelism, Scalability, Programmability*. McGraw-Hill, Inc., New York, 1993.

[41] IBM. The CoreConnect™ Bus Architecture. available at http ://www.chips.ibm.com/product/coreconnect/docs/crcon_wp.pdf, 1999.

[42] IDT. IDT Peripheral Bus (IPBus^{TM}) Intermodule Connection Technology Enables Broad Range Of System-Level Integration. white paper.

[43] intel. *Intel Architecture, Software Developer's Manual, Volume 2 : Instruction Set Reference*. Order number : 243191.

[44] International Trend Roadmap on Silicon. *System-on-Chip survey*, 2001. available at http ://www.itrs.org/.

[45] T. B. Ismail, J. Daveau, K. O'Brien, and A. A. Jerraya. A system-level communication synthesis approach for hardware/software systems. *Microprocessors and Microsystems*, 20(3) :149–157, May 1996.

[46] D. Jaggar. *Advanced RISC Machines Architectural Reference Manual*. Prentice Hall, July 1996.

[47] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proc. of the 21st Int'l Symposium on Computer Architecture*, 1994.

[48] O. Kyas. *ATM networkd*. Thomson Publishing, 1995.

[49] B. Lee and A. R. Hurson. Dataflow architectures and multithreading. *IEEE Computer*, pages 27–39, Aug. 1994.

[50] J. A. J. Leijten et al. PROPHID : A Heterogeneous Multi-Processor Architecture for Multimedia. In *Proc. Int'l Conference on Computer Design*, 1997.

[51] C. K. Lennard, P. Schaumont, G. de Jong, A. Haverinen, and P. Hardee. Standards for System-Level Design : Practical Reality or Solution in Search of a Question ? In *Proc. Design Automation and Test in Europe*, Mar. 2000.

[52] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The dash prototype : Logic overhead and performance. *IEEE Transactions on Processor Design Symposium*, 1993.

[53] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya. Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip. In *Proc. Design Automation Conf.*, pages 518–523, 2001.

[54] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya. An Optimal Memory Allocation for Application-Specific Multiprocessor System-on-Chip. In *Proc. Int'l Symposium on System Synthesis*, Oct. 2001.

[55] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya. Automatic Code-Transformations, and Architecture Refinement for Application-Specific Multiprocessor SoCs with Shared Memory. In *11th IFIP International Conference on Very Large Scale Integration*, Dec. 2001.

[56] Mentor Graphics, Inc. Seamless CVE. available at http ://www.mentorg.com/seamless/ .

[57] MHS electronic. *SPARC RISC USER'S GUIDE*.

[58] M. M. Michael, A. K. Nanda, B.-H. Lim, and M. L. Scott. Coherence controller architectures for SMP-based CC-NUMA multiprocessors. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture (ISCA'97)*, pages 219–228, 1997.

[59] L. Moll and M. Shand. Systems performance measurement on pci pamette. In *a determiner*, 1997.

[60] M. D. Nava and C. Del-Toso. A short overview of the vdsl system requirements. *IEEE Communications*, 40(12) :82–90, Dec. 2002.

[61] M. D. Nava and G. S. Ökvist. The zipper prototype : A complete and flexible vdsl multicarrier solution. *IEEE Communications*, 40(12) :92–105, Dec. 2002.

[62] G. Nicolescu, S. Martinez, L. Kriaa, W. Youssef, S. Yoo, B. Charlot, and A. A. Jerraya. Application of multi-domain and multi-language cosimulation to an optical mem switch design. In *Proc. Asia South Pacific Design Automation Conference*, Jan. 2002.

[63] G. Nicolescu, S. Yoo, and A. A. Jerraya. Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design. In *Proc. Design Automation and Test in Europe*, Mar. 2001.

[64] E. G. Nuta Nicolescu. *Spécification et validation des systèmes hétérogènes embarqués*. PhD thesis, Laboratoire des Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs - Intstitu National Polytechnique de Grenoble, Grenoble, France, 2002.

[65] J. Oberg. *ProGram : A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols*. PhD thesis, Department of Electronics, Electronic System Design, Royal Institute of Technology, Electrum 229, Isafjordsgata 22–26, S-164 40 Kista, Sweden, 1999.

[66] V. Oksman. *Standard VDSL Technology*. Broadcom corp. available at : httpd ://www.ieee802.org/3/efm/public/jul01/presentations/oksman_1_0701.pdf.

[67] M. O'Nils. *Specification, Synthesis and Validation of Hardware/Software Interfaces*. PhD thesis, Department of Electronics, Electronic System Design, Royal Institute of Technology, Electrum 229, Isafjordsgata 22–26, S-164 40 Kista, Sweden, 1999.

[68] M. O'Nils and A. Jantsch. Operating System Sensitive Device Driver Synthesis from Implementation Independent Protocol Specification. In *Proc. Design Automation and Test in Europe*, Mar. 1999.

[69] Open Microprocessor System Initiative. *PI-Bus VHDL Toolkit*. Version 3.1.

[70] OpenCore Protocol International Partnership. *OpenCore Protocol*. available at http ://www.ocpip.org/home.

[71] PC/104 Consortium. *PC/104-Plus Specification*, June 1997. Version 1.1.

[72] PCI SIG. *PCI Conventional specification, release 2.3*.

[73] J. L. Peterson and A. Silberschatz. *Operating System Concepts*. Addison-Wesley Publishing Co. Inc., 2nd edition.

[74] Philips Semiconducteur. *Home Entertainment Engine - Nexperia pnx8500*.

[75] Qualcomm, Inc. *MSM3300*. available at http ://www.qualcomm.com/cdmatechnologies/products/msm3300.html.

[76] Qualcomm, Inc. *CDMA System Engineering Training Handbook*, 1993.

[77] RASSP Taxonomy Working Group (RTWG). VHDL Modeling Terminology and Taxonomy, May 1999. available at http ://www.atl.external.lmco.com/rassp/taxon/rassp_taxon.html.

[78] Rational inc. *UML*, 2002. URL : http ://www.rational.com/uml/.

[79] J. A. Rawson and al. Interface Based Design. In *Proc. Design Automation Conf.*, 1997.

[80] A. Seawright. *Grammar-based specifications and synthesis for synchronous digital hardware design*. PhD thesis, Computer Science Dept., Univ. of California, Santa Barbara, 1994.

[81] R. Seepold, L. Rosenberg, M. Genoe, and G. Matthew. SPECIAL SESSION - Virtual Socket Interface Alliance. In *Proc. Design Automation and Test in Europe*, 1999.

[82] Silicore Corporation. *Application Note : WBAN001 WISHBONE Interface For Slave I/O Ports*, 1999.

[83] Silicore Corporation. *Application Note : WBAN002 WISHBONE Interface For Memory Elements*, 1999.

[84] Silicore Corporation. *Application Note : WBAN003 Design Philosophy of the WISHBONE SoC Architecture*, 1999.

[85] Silicore Corporation. *Wishbone Interconnection for Portable IP Cores Specification Revision A*, 1999.

[86] Sonics, Inc. *Sonics μNetworks, Technical Overview*, June 2000. Networking Technology Datasheets, available at http ://www.sonicsinc.com/Documents/Overview.pdf.

[87] Sun microsystems, Inc. *MAJC Architecture Tutorial*. available at http ://www.sun.com/microelectronics.

[88] Synopsys. *Protocol Compiler User's Manual*, 1998. Mountain View, California.

[89] Synopsys Inc. *Design Compiler Family*. available at http ://www.synopsys.com/products /logic/design_comp_cs.html.

[90] Synopsys, Inc. *SystemC, Version 1.1*. available at http ://www.systemc.org/.

[91] Synopsys, Inc. *SystemC, Version 2.0*. available at http ://www.systemc.org/.

[92] Synopsys Inc. *VHDL System Simulator*. available at http ://www.synopsys.com/products /simulation/vss_cs.html.

[93] H. Tago. CPU for PlayStation 2. In *Workshop on Synthesis and System Integration of Mixed Technology (SASIMI)*, 2000.

[94] K. Tammemae, M. O'Nils, A. Jantsch, and A. Hemani. Akka : a toolkit for cosynthesis and prototyping. In *IEE Digest No.96/036 of Colloquium on Hardware-software Cosynthesis for Reconfigurable Systems*, pages 8/1–8/8, 1996.

[95] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall Inc., 1992.

[96] A. Tanenbaum. *8.1.5 Taxonomie des ordinateurs parallèles*, pages 540–542. Dunod, 4 edition, 1999. traduction de [97].

[97] A. Tanenbaum. *Structured Computer Organization*. Prentice Hall Inc., 4 edition, 1999.

[98] M. Tremblay. MAJC : Microprocessor Architecture for Java Computing. In *HotChips*, 1999.

[99] M. Tremblay. MAJC-5200 : A VLIW Convergent MPSOC. In *Microprocesseur Forum*, 1999.

[100] M. Trenas, J. Lopez, and E. L-Zapata. A memory system supporting the efficient simd computation of the two dimensional dwt. In *ICASSP '98*, May 1998.

[101] A. Turing. On computable numbers, with an application to the entscheidungsproblem. In *London Mathematical Society*, volume 42 of *2*, 1936. reprinted in M. David (ed.), The Undecidable, Hewlett, NY : Raven Press, 1965.

[102] F. Vahid and D. Gajski. Slif : A specification-level intermediate format for system design. In *Proc. European Design & Test Conf.*, pages 116–123, Mar. 1995.

[103] F. Vahid and T. Givargis. Integrating Cores into System-Level Specification. In *Proc. Int'l Symposium on System Synthesis*, Dec. 1998.

[104] C. A. Valderrama, A. Changuel, P. V. Vijaya-Raghavan, M. Abid, T. Ben Ismail, and A. A. Jerraya. *A unified model for co-simulation and co-synthesis of mixed hardware/software systems*, pages 579–583. Morgan Kaufmann Publishers, 2001.

[105] S. Vercauteren. *Hardware/Software Co-Design of Application Specific Heterogeneous Architectures*. PhD thesis, Katholieke Universiteit Leuven, Fakulteit Toegepaste Wetenschappe, Departement Elektrotechniek (ESAT), Divisie INSYS, Kard. Mercielaan 94, B-3001 Leuven, België, Dec. 1998.

[106] S. Vercauteren, B. Lin, and H. de Man. Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications. In *Proc. Design Automation Conf.*, June 1996.

[107] S. Vercauteren, J. van der Steen, and D. Verkest. Combining software synthesis and hardware/software interface generation to meet hard real-time constraints. In *a determiner*, 1999.

[108] M. H. Weiss and G. P. Fettweis. A new scalable DSP architecture for mobile communication applications. In *Australasian Computer Architecture Conference, Auckland, New Zealand*, pages 149–160. Springer-Verlag, Singapore, 1999.

[109] W. Wolf. *Computers as Compunents : Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers, 2001.

[110] W. Ye, R. Ernst, T. Benner, and J. Henkel. Fast timing analysis for hardware-software cosynthesis. In *Proc. IEEE Int. Conf. on Computer Design*, pages 452–457, 1993.

[111] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, and A. A. Jerraya. A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design. In *Proc. Int'l Workshop on Hardware-Software Codesign*, Apr. 2001.

Glossaire

A

ADC Analog to Digital Converter.
Convertisseur analogique vers numérique : transformation d'une grandeur continue en une représentation numérique quantifiée. Le quantum utilisé lors de cette transformation est appelé la résolution : $resolution = \frac{Amplitude_{max\ du\ signal\ continue}}{nombre\ de\ valeurs\ possible\ du\ signal\ numérique}$.

ADSL Asynchronus Digital Subscriber Line.
Liaison DSL entre des terminaisons de débits différents. Un terminal peut envoyer des données vers un serveur jusqu'à 800Kbps et en recevoir jusqu'à 8Mbps.

AHB AMBA High-Bandwidth.
Version à bande passante élevée du bus AMBA.

AICP Advanced Image Composition Processor.
Processeur matériel embarqué dans le *Viper* de *Philips* pour accélérer les compositions en mosaïques et la surimpression d'images.

AL Architecture Locale.
Groupement de composants de calcul, de mémorisation et de communication s'auto-suffisant à la réalisation modulaire de fonctionnalités. Ils se composent généralement d'un ou plusieurs processeurs de logiciel, de mémoires contenant les instructions (réalisation des fonctionnalités logicielles) et les données locales, de composants matériels d'accélération et d'un bus partagé interconnectant tout ce petit monde.

AMBA Advanced Multi-masters Bus Architecture.
Standard propriétaire de bus, développé par ARM corp.

API Application Program Interface.
Interface logicielle permettant à une application d'accéder aux services proposés par le système d'exploitation. Une API, matérialisée par un ensemble de prototype de fonctions systèmes, fournit une abstraction de l'architecture matérielle aux différentes tâches logicielles, en facilitant ainsi la portabilité.

ASIC Application Specific Integrated Circuit.
Circuit (ou portion de circuit) intégré développé spécifiquement pour une Application.

ASAG Application-Specific Architecture Generation.
Outil de génération d'architecture spécifique à une application. Cet outil et les bibliothèques qui lui sont associées ont été développés dans le cadre de ces travaux.

ASCII American Standard Code for Information Interchange.
Représentation standard sur 7 ou 8 bits des caractères.

ASOG Application-Specific Operating-Systems Generation.
Outil de génération de système d'exploitation spécifique à une application. Cet environnement de conception logicielle est le fruit de [29].

ATM Asynchronous Transfert Mode.
Technologie permettant l'implémentation du réseau RNIS large bande.

B

BCA Bus Cycle Accurate.
Niveau d'abstraction pour la co-conception utilisé par l'outil N2C de CoWare. Il est équivalent au « **Detailed Behavioral** » pour les parties matérielles et au « **Object Code** » pour les parties logicielles, tous deux définis par [77].

BCASH Bus Cycle Accurate Shell.
Pseudo-niveau d'abstraction utilisé par l'outil N2C de CoWare. Il est issu de l'union des niveaux BCA et UT. Il est destiné à la modélisation d'interface d'abstraction entre les niveaux BCA et UT.

BCU Bus Control Unit.
Unité de contrôle d'un bus de communication. Elle embarque le comportement de l'arbitre du bus, mais aussi du décodeur afin de pouvoir mettre en œuvre des polices d'attribution complexes.

BFM Bus Functional Model.
Modèle fonctionnel du bus d'un processeur, que l'on utilise pour coupler un ISS à son environnement matériel.

BIST Built-In Self Test.
Matériel additionnel et autonôme, intégré dans les mémoires afin d'en tester le bon fonctionnement et de procéder à d'éventuelles réparation par redirection sur des unités de secours.

bit binary digit.
Quantité d'information pouvant être exprimée par un variable binaire.

bps bits par seconde.
Unité de débit employée pour la mesure de performance des communications numériques. Les anglo-saxons lui associent le terme « baud-rate ».

C

CA Chanel Adaptor.
Unité matérielle utilisé pour contrôler les communications d'un nœud de calcul au travers un canal. Cette unité est spécifique au protocole du canal, sa bande passante et quelques autres particularités sont réglées pour répondre aux besoins de l'application.

CC Coprocesseur de Communication.
Accélérateur matériel contrôlant toutes les communications d'un nœud de calcul avec l'extérieur.

CC-NUMA Coherent Cache Non Uniform Memory Access.
Sous familles des architectures NUMA utilisant des caches dont les contenus sont maintenus cohérents par l'utilisation de mécanisme de commuication, diffusion de très faible latence.

CDMA Code Division Multiple Access.
The spread spectrum modulation used in the Qualcomm system.

CDFG Control & Data Flow Graph.
Extension du modèle DFG pour supporter des concepts de contrôle dans le contexte de la co-conception logicielle et matérielle.

Cedit Colif Editor.
Outil d'édition graphique de descriptions Colif.

CISC Complex Instruction Set Computer.
Famille d'architecture de processeur ayant un jeu évolué et conséquent d'instructions. L'exécution de ces instructions peuvent requérir un nombre variable, pouvant atteindre plusieurs dizaines, de cycles horloges. Cette architecture est délaissée par les systèmes embarqués au profit de l'architecture RISC. On trouve plusieurs formulations pour expliquer la signification du sigle « CISC », en français : « à jeu (ou ensemble) complexe d'instructions », « à jeu (ou ensemble) complet d'instructions », « à jeu (ou ensemble) d'instructions complexes ». En fait, c'est le jeu (ou l'ensemble) qui est complexe (ou complet) et non les instructions elles-mêmes. En effet, le jeu compte des instructions plus nombreuses que dans un ordinateur à microprocesseur dont

l'architecture est de type RISC (« Reduced Instruction Set Computer »). C'est pour cette raison que l'accord de l'adjectif « complexe » ou « complet » doit se faire avec le terme « jeu » (ou « ensemble ») plutôt qu'avec le terme « instructions » (féminin pluriel). « CISC » et « RISC » (« Reduced Instruction Set Computer ») font référence à deux technologies qui s'opposent.

Colif COdesign Language Independant Format.
Langage de composition de systèmes hétérogènes en niveaux d'abstraction, en protocoles de communication et en langages de description.

COMA Cache Only Memory Access.
Famille d'architecture multi-processeurs dont chaque processeur accède à une mémoire partagée e tcentralisée, au travers d'une mémoire cache privée.

CORBA Common Object Request Broker Architecture.
Standard définissant l'architecture que doivent adopter les systèmes d'exploitation et les applications pour rendre possible la communication entre des objets provenant d'environnements différents. Le standard Corba n'est pas une norme au sens strict du terme, car ses spécifications n'ont pas été entérinées par un organisme officiel de normalisation. On le considère comme une norme de facto (ou norme de fait), car il a été défini et adopté de façon consensuelle par un ensemble d'entreprises regroupées sous le nom de Object Management Group ou OMG. L'application du standard Corba permet de recréer, dans Internet, l'architecture client-serveur distribuée. Le mécanisme autorisant les objets à échanger des messages et à s'activer mutuellement est appelé gestionnaire d'objets distribués ou Object Request Broker (ORB). Les termes architecture Corba, technologie Corba et modèle Corba font référence à des notions plus spécifiques. La traduction littérale de l'expression Common Object Request Broker Architecture est : architecture pour un intermédiaire commun dans les requêtes d'objet.

CMOS Complementary Metal-Oxyd Semiconductor.
Technologie de fabrication de circuits intégrés qui associe une paire de transistors complémentaires (l'un de type N, l'autre de type P) sur un même support. Les circuits intégrés obtenus consommant peu d'énergie, ils sont souvent utilisés pour les mémoires et les microprocesseurs des ordinateurs portables. Toutefois, ils sont fort sensibles à l'électricité statique.

CPU Central Processing Unit.
Le sigle CPU correspond au terme anglais central processing unit. Il est utilisé pour désigner l'unité centrale de traitement d'un ordinateur, c'est-à-dire son processeur principal.

Cview Colif Viewer.
Outil d'affichage graphique de descriptions Colif.

D

DAC Digital to Analog Converter.
Convertisseur de données numériques en une grandeur analogique pouvant être continue.

DASH Directory Architecture for SHared memory.
Projet d'architecture multi-ordinateurs (multi-clusters) de l'université de Standford.

DCT Digital Celsius Transfert.
Transformée numérique de Celsius. Très similaire à la transformée de Fourrier, mais utilise des coefficients différents.

DDRAM Double port Dynamic Random Access Memory.
Famille de mémoire dynamique ayant deux points d'accès autonomes et concurrents.

DES Data Encryption Standard.
Standard d'encryptage de données basé sur des opérations de décalage et d'entrelaçage des données et d'une clé de codage.

DFG Data Flow Graph.
Représentation du comportement d'un élément sous la forme d'un graphe. Les sommets de ce graphe sont des taches concurrentes. Les arcs matérialisent les dépendances de données inter-taches (Producteur $\longrightarrow$ Consommateur). Ce modèle est issu de la communauté logicielle.

DMA Direct Memory Access.
Unité matérielle dynamiquement configurable opérant des recopies de blocs mémoire à partir d'une adresse « source » vers une adresse « destination ».

DMT Discrete MultiTone modulation.
Modulation d'amplitude quadratique.

DPRAM Dual-port Random Access Memory.
cf. DDRAM.

DRAM Dynamic Random Access Memory.
Famille de mémoire à technologie dite « dynamique ». Les points de mémorisation sont réduits à des transistors dont la capacité de grille est suffisante à une rétention momentanée de l'information. Ce sont les courants de fuite au travers le substrat qui rendent cette mémorisation temporaire et nécessite le rafraîchissement périodique de l'information par lecture-réécriture.

DSHM Distributed Shared Memory.
Ressources de mémorisation distribuées géographiquement sur une architecture afin d'offrir des accès priviligiés à quelques composants de calcul.

DSP Digital Signal Processor.
Processeur spécialisé pour le calcul d'algorithmes de traitement du signal.

DSL Digital Subscriber Line.
Technologie de communication par paire torsadée, utilisée pour les liaisons numériques RNIS.

DTD Data Type Document.
Fichier de définition de structure de donnée permettant l'extension de la syntaxe du langage XML.

E

ECO Engineer Change Order.
Requête émise par les ingénieurs d'implémentation vers les concepteurs afin de contourner une difficulté technologique par une modification des spécifications.

EDF Earliest-Deadline First.
Police d'élection basée sur des priorités dynamiques inversement proportionnelles au délai restant à une fonctionnalité/tâche/communication pour être opérée.

EDRAM Embedded Dynamic Random Access Memory.
Famille de mémoire dont la technologie permet l'intégration sur la même puce que le processeur.

EEPROM Electricaly Erasable, Programable and Read-Only Memory.
Sous famille de mémoire EPROM écrivable et effaçable électriquement.

EFU Elementary Functional Unit.

EPROM Erasable, Programable and Read-Only Memory.
Sous famille de mémoire ROM effaçable par exposition aux ultra-violets et écrivable par l'utilisation de tensions élevées ($\tilde{1}5$ V).

F

FDIV Floating Divider.
Processeur matériel de division de nombres flottants.

FFT Fast Fourrier Transfert.
Algorithme rapide de calcul de transformée de Fourrier s'appliquant avec 2^n points, connu sous le nom d'algorithme « papillon » (« Butterfly »).

FIFG First-In First-Granted.
Police d'élection basée sur le principe : « Premier arrivé (ou pret), premier servi ».

FIFO First-In First-Out memory.

FLASH Flexible Architecture for SHared memory.
Architecture multi-processeurs NUMA.

FLASH Floating gate transistor.

FMAC Floating Multiplier-Accumulator.
Processeur matériel de multiplication et d'accumulation de nombres flottants.

FPGA Field Programmable-Gates Array.

FPIC Field Programmable Interconnect Circuit.

FPU Floating Processor Unit.

FRAM Ferro-magnetic Random Access Memory.

FSM Finite State Machine.

FU Functional Unit.

G

GFLOPS Giga Floating-Operation Per Second.
Unité de mesures de performances de processeur en termes de calcul en nombres flottants. Elle quantifie ces mesures en Milliards d'opération à la seconde.

GPCPU General Purpose Central Processing Unit.
Processeur logiciel générique ne possédant pas de ressources de calcul spécifiques apte à l'accélération de l'exécution d'algorithmes particuliers (traitement du signal, etc.).

GSM Global System for Mobile communication.
GSM est un standard Européen de communication numérique et cellulaire.

H

HAL Hardware Abstraction Layer.
Couche de logiciel interfaçant des fonctionnalités génériques aux ressources matérielles, les rendant ainsi indépendantes de l'architecture (portable).

HDL Hardware Description Language.

HDSL Higth-speed DSL.
Standard de communication appartenant à la famille xDSL.

HLL High-Level Language.
Famille de langages standards de spécification logicielle permettant une modélisation de fonctionnalités indépendamment de l'architecture matérielle d'exécution.

HLS High-Level Synthesis.
Synthèse d'une description algorithmique d'un composant matériel en une représentation d'implémentation.

HW Hardware.
Terme anglo-saxon couramment employé pour désignée les composantes matérielles et faiblement flexible de l'architecture d'un système.

Hz Hertz.
Unité de mesure de fréquence. 1 Hz = Fréquence d'un phénomène périodique dont la période est de une seconde.

I

IB Internal Bus.

IP Intellectual Property.
Partie centrale d'un processeur, dont la réutilisation pour fabriquer de nouveaux circuits est protégée par les règles de la propriété intellectuelle. Les termes coeur IP et bloc IP ont été construits en utilisant l'abréviation anglaise du terme intellectual property. Les termes bloc fonctionnel et composant virtuel (angl. virtual component) sont parfois employés pour définir le coeur de propriété intellectuelle.

IPBus Integrated Peripherals Bus.

ISA Instruction Set Architecture.
Niveau d'abstraction matériel servant à caractériser un processeur programmable.

ISDN Integrated Service Digital Network.
Voir RNIS.

ISO International Standard Organization.
Organization internationale validant officiellement des standards.

ISS Instruction Set Simulator.
Simulateur d'un processeur (ou jeu d'instruction) permettant de valider une description logicielle au niveau code objet.

J

JTAG Joint Test Action Group.
Standard pour fournir un accès externe et sériel au test intégré dans un circuit, via 5-broches. Le standard JTAG a été adopté comme standard IEEE (IEEE 1149 Standard Test Access Port and Boundary-Scan Architecture).

L

LIFO Last In First Out memory.
Mémoire ayant un seul point d'accès en lecture et écriture. Son fonctionnement se base sur un mécanisme de pile (qui est d'ailleurs son nom Français). Ainsi les accès en lecture sur une LIFO renvoient les données dans l'ordre anti-chronologique de leur écriture.

LCP Langage Clair et Précis.
Utilisation d'un sous-ensemble d'une langue vivante telle que le Français ou l'Anglais pour modéliser une fonctionnalité.

K

Kbps Kilo-bits par seconde.
Multiple de l'unité de débits pour les communications numériques : le bps. 1 $Kbps = 10^3\ bps$.

KHz Kilo-Hertz.
Multiple de l'unité de fréquence : le Hz. 1 $KHz = 10^3\ Hz$.

M

MA Module Adaptor.

MAC Multiplier-Accumulator.
Processeur matériel réalisant la multiplication de deux nombres et accumulant le résultat.

Mbps Méga-bits par seconde.
Multiple de l'unité de débits pour les communications numériques : le bps. 1 $Kbps = 10^6\ bps$.

MCU Micro-Controler Unit
. Processeur ayant des performances, une consommation électrique et une occupation surfacique toutes modérées. Ils sont par conséquent fortement utilisés dans des applications embarquées où ils ont pour charges de contrôler des accélérateurs matériels.

MEM Mémoire.

MEMS Micro Electrical and Mechanical System.

MHz Mega-Hertz.
Multiple de l'unité de fréquence : le Hz. 1 $MHz = 10^6\ Hz$.

Middle Mark-up internal data-description language extension.
Adjonction au langage XML sous la forme d'un DTD afin de définir de nouvelles structures de données telles que COLIF.

MIMD Multiple Instruction streams, Multiple Data streams.
Type d'architecture parallèle. Dans cette configuration, on découpe le programme informatique en tâches différentes et chacune de celles-ci est affectée à un seul processeur.

MISD Multiple Instruction streams, Single Data streams.

modem modulateur-démodulateur.
Partie fonctionnelle d'une terminaison de communication, en charge d'implémenter la couche physique du protocole par modulation/démodulation de fréquence du signal porteur.

MMI Memory Management Interface bus.
Réseau de connexions point-à-point reliant une mémoire partagée à plusieurs clients. Ce bus est la colonne vertébrale du *Viper* de *Philips*.

MMU Memory Management Unit.
Unité matériel insérée entre un composant de calcul et la mémoire, afin de translater de façon transparente les addresses de cette dernière.

MMR Memory-Mapped Register.
Registre de configuration ou d'états accessible comme un emplacement mémoire conventionnel assigné à une adresse donnée.

MPEG Moving Picture Experts Group.
Standard de compression de flux vidéo.

MSP MPEG System Processor.
Processeur de traitement de flux vidéo utilisé par le *Viper* de *Philips*.

N

N2C Napkin-to-Chip.

NC-NUMA No Cache Non Uniform Memory Access.
Sous familles d'architectures NUMA dont les accès mémoires ne sont pas cachés.

NoC Network-on-Chip.
Système monopuce organisé comme un réseau de processeur.

NUMA Non Uniform Memory Access.
Famille d'architectures multi-processeurs dont la mémoire est partagée est distribuée sur toute l'architecture afin de privilégier l'accès d'un processeur donné à un composant mémoire donné.

O

OCP Open Core Protocol.

ORB Object Request Broker.
« Intergiciel respectant les spécifications de la norme CORBA, qui régit l'échange des messages et des services entre les objets distribués d'une application client-serveur. Le gestionnaire ORB rend transparent l'accès à des objets issus d'environnements différents dans le réseau. ». D'après l'Office de la langue Française.

OS Operating System.
Un Operating System ou systéme d'exploitation, est une couche logicielle permettant d'abstraire une architecture matérielle à la vue du logiciel applicatif afin de gérer implicitement l'utilisation partagée des ressources, mais aussi de garantir la portabilité de l'application sur des architectures diverses.

OSI Open System Interconnect.
Représentation de protocoles de communication standardisée par l'ISO.

P

PC Personal Computer.
Ordinateur à usage personnel (sous-entendu un unique utilisateur), par opposition au « Mainframes ».

PCA Processor Centric Architecture.

PCI Peripheral to Central Processing Unit Interconnects.

PIbus Peripheral Interconnect bus.
Bus partagé multimaître développé par OMI.

PIC Peripheral Interrupts Controler.
Contrôleur de requêtes d'interruption émises par des périphériques en direction du processeur. Il en permet le multiplexage orthogonalement à l'application de priorités et de masquage. Il peut aussi avoir à gérer l'acquittement de ces requêtes.

PLL Phase-Locked Loop.

PSP Processor Support Packages.

PU Process Unit.

pipeline .
Séquence d'unités fonctionnelles réalisant une tâche en plusieurs tranches, telle une chaîne de montage en usine automobile.

R

RAM Random Access Memory.

RF Radio-Fréquence.
Domaine de la conception électronique relatif aux applications communicantes par voies hertziennes à des fréquences élevées.

RISC Reduce Instruction Set Computer.
Famille d'architecture de processeur ayant un jeu simplifié et réduit d'instructions. Bien que leurs instructions soient moins puissantes que celles de la famille CISC, l'utilisation de processeur RISC est répandue dans les applications embarquées en raison de leur plus faible consommation ; De plus une meilleure prédiction des temps d'exécution de leurs instructions permet de vérifier leur tenue aux contraintes « temps-réel » de l'application.

Rive Rive.
Langage de macro complet permettant la génération de chaîne de caractères au sein d'un ensemble de texte statique. Rive est issu de [29].

RMA Rate Monotonic Analysis.
Méthode d'attribution de priorités statiques à des fonctionnalités se partageant des ressources. Si chaque unité réclame avec une fréquence connue et immuable, l'accès aux ressources partagées, alors l'assignation de priorités proportionnelles à ces fréquences permet de garantir un respect des échéances de chacune de ces

fonctionnalités. Le taux d'utilisation des ressources $\Gamma = \sum \frac{\tau_i}{T_i}$ doit être tel que : $\Gamma < n(2^{\frac{1}{n}} - 1)$, avec τ_i le délai d'utilisation des ressources par la fonctionnalité n$^o i$, T_i la période de cette utilisation, et n le nombre de fonctionnalités se partageant ces ressources. L'ordonnancement des accès aux ressources s'appelle alors RMS.

RMS Rate Monotonic Scheduling.
Ordonnancement de l'utilisation de ressources partagées basé sur des priorités statiques assignée selon la méthode RMA.

RNIS Réseau Numérique à Intégration de Service.

ROM Read-Only Memory.

RPC Remote Procedure Call.

RTC Register Transfert C.

RTL Register Transfert Level.

S

SAP Service Access Port.

SISD Single Instruction stream, Single Data stream.

SIMD Single Instruction stream, Multiple Data streams.
Machine à structure massivement parallèle capable de traiter une instruction sans affectation préalable des processeurs de traitement.

SDL System Description Language.

SDRAM Synchronous Dynamic Random Access Memory.

SDSL Symmetric DSL.
Standard de communication par modulation de fréquence multiporteuse appartenant à la famille xDSL.

SE Système d'Exploitation.
c.f. OS.

SHM Shared Memory.
Ensemble de ressources de mémorisation à usage (ou accès) partagé. Ce terme recouvre aussi, par abus de langage, la notion de centralisation de ces ressources en complète opposition aux DSHM.

SLS System Level Synthesis.

SMP Symetric Multi-Processor.

SoC System-on-Chip.
«Système monopuce » : Circuit intégrant sur une même puce, divers éléments fonctionnels tels que des processeurs, des mémoires, des accélérateurs matériels, ...

SPRAM Single-Port RAM.
Bloc de mémoire vive ayant un unique port logique d'accès.

SPW Cadence Signal Processing Worksystem.

SRAM Static Random Access Memory.

SW Software.

T

TAP Test Access Port.

TDMA Time Division Multiple Access.
Allocation d'une ressource de calcul ou de communication par tranches de temps. Toutes les unités utilisant ainsi des ressources, ne peuvent y accéder que cycliquement, pour un nombre de tranches de temps et dans un ordre statiquement définis.

TIMA Techniques Informatiques et Microélectroniques pour l'Architecture des ordinateurs.

U

UAL Unité Arithmétique et Logique.

UCOM Communication Unit.
Adaptateur de communication généré par l'outil GAUT[4].

UMA Uniform Memory Access.
Famille d'architectures multi-processeurs contenant une mémoire centralisée accessible par tous les processeurs de façon uniforme tant en coûts, qu'en performances.

UML Unified Modeling Language.
Langage de modélisation par objets de troisième génération, permettant de déterminer et de présenter les composants d'un système objet lors de son développement, ainsi que d'en générer la documentation. En présentant une description unifiée des concepts objets, le langage UML fait franchir le pas menant de la technologie des objets à la technologie des composants. Le langage UML est le résultat de la fusion des travaux de Rumbaugh, Booch et Jacobson.

UT UnTimed.

V

VADeL Virtual Architecture Description Language.
Langage de composition de systèmes hétérogènes, basé sur une extension de [90].

VCA Virtual Chanel-Adaptor.
Adaptateur virtuel de canal permettant de spécifier la connexion d'un CA dans une PCA générique. Ce composant virtuel sera dupliqué et spécialisé lors de la génération de la PCA cible (cf. sous section 4.1.2).

VCC Virtual Component Codesign.
Outil d'exploration c'architecture commercialisé par Cadence inc.

VCI Virtual Component Interface.
Modèle standard d'interface élaboré par VSIA.

VDSL Very hight data rate Digital Subscriber Line.
Standard de communication par paire torsadée.

VHDL Very high-scale integrated Hardware Description Language.
Langage de description de système électronique numérique, basé sur une extension de Ada.

VHLL Very-High-Level Language.
Langage de description de fonctionnalité de très haut niveau indépendant de toutes hypothèses de réalisation. UML et SDL sont de bons exemples d'un tel concept.

VLIW Very Long Instruction Word processor.
Architecture de processeur à grand nombre de FUs. Une instruction de tels processeur est construite par l'assemblage de sous-instructions dédiées à une FU. Le parallélisme de l'application exécutée doit donc être extrait de façon statique lors de l'étape de compilation.

VSIA Virtual Socket Interface Alliance.
Consortium d'industriels œuvrant pour la définition d'un standard de modélisation des interfaces de composants matériels.

W

WCDMA Wide Code Division Multiple Access.
Protocole de communication pour téléphone cellulaire, issue d'une extension de CDMA.

X

XML Extensible Mark-up Language.
Langage à balises, extensible par l'adjonction d'un modèle de document (le DTD) définissant les nouvelles structures de données.

xDSL extended Digital Subscriber Line.
Famille de standards de communication par paire torsadée, basée sur des extensions de DSL.

Index

Index

RÉSUMÉ

Cette thèse adresse une recherche d'automatisation pour la conception d'architecture matérielle de systèmes monopuces.
L'accroissement incessant de la complexité des systèmes, l'amincissement des fenêtres commerciales et la réduction du temps accordé à la conception qui en résulte apportent une divergence non résolue entre les besoins en productivité et celle effective des équipes de concepteurs. Une réponse est proposée par l'abstraction de ces systèmes jusqu'à un niveau où les caractéristiques de l'architecture matérielle ne sont que des directives d'implémentation annotées à un modèle purement fonctionnel de l'application. Ce modèle est alors pris en charge par un flot d'étapes de raffinement automatisées, le conduisant jusqu'à un niveau de détail permettant l'utilisation d'outils de synthèse matérielle commerciaux qui, à leur tour, le transposent en silicium.
La contribution de cette thèse à cette méthodologie concerne la définition d'une représentation de ces architectures matérielles à chaque niveau utilisé au cours de la conception, ainsi qu'une approche d'automatisation du flot de raffinement par l'assemblage systématique de composants de bibliothèques. La pertinence de ces travaux a été évaluée par leurs applications à la conception de plusieurs systèmes dont un modem VDSL et un terminal GSM WCDMA présentés dans ce mémoire.

MOTS-CLÉS

Système monopuce, multiprocesseur, hétérogénéité, conception à base de composants, assemblage systématique, CAO, architecture spécifique à l'application.

TITLE

An approach for the systematic gathering of interface items toward the generation of multiprocessor architectures

ABSTRACT

This thesis tackles a research toward an automation in design of System-on-Chip (SoC) hardware-architectures.
The everlasting gap moving a design-team's productivity away from claims brought by the endless rising complexity of systems, thus combined with thinner and thinner market windows and related shortened design cycles, still applies while dealing with SoC design. The elevation of the abstraction-level, used to model systems, up-to pure functionalities annotated with architecture design-directives is the aimed solution. That model would be refined down-to a detailed and more silicon-compliant level. Commercial hardware-synthesis tools would then be able to achieve the final translation toward silicon.
This thesis contributes in two points to that method. The former addresses the definition of a representation model of SoC hardware-architectures mandatory while diving through the abstraction-levels and the design-steps. The latter provides an approach to the automation of this hardware-architecture refinement flow through the systematic gathering of library-components. The relevance of that work is evaluated thanks to several case-studies among them a VDSL modem and a WCDMA cellular phone are introduced in this report.

KEYWORDS

System-on-Chip, multiprocessor, heterogeneity, component-based design, systematic gathering, CAD tool, application-specific architecture.

INTITULÉ ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix VIALLET, 38031 GRENOBLE CEDEX, FRANCE.

ISBN : 2-913329-93-4 ISBNE : 2-913329-94-2