Synthèse des interfaces de communication dans la conception des systèmes monopuces : de la spécification à la génération automatique

A. Grasset

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|__/__/__/__/__/__/__/__/__/__|

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de l'Ecole Doctorale

"Electronique, Electrotechnique, Automatique, Télécommunication, Signal"

présentée et soutenue publiquement par

Arnaud GRASSET

le 4 Janvier 2005

Synthèse des interfaces de communication dans la conception des systèmes monopuces : de la spécification à la génération automatique

Directeur de thèse
Ahmed Amine Jerraya
Co-encadrant de thèse
Frédéric Rousseau

Jury :

M. Pierre GENTIL	**, Président**
M. Jean Luc PHILIPPE	**, Rapporteur**
M. Bruno ROUZEYRE	**, Rapporteur**
M. Ahmed Amine JERRAYA	**, Directeur de thèse**
M. Frédéric ROUSSEAU	**, Co-encadrant**

Remerciements

Je voudrais sincèrement remercier toutes les personnes qui ont contribué à l'élaboration de cette thèse par leur aide, leurs conseils et leur soutien.

J'adresse tout d'abord mes remerciements à Monsieur Ahmed Amine Jerraya, directeur de recherche au CNRS et chef de l'équipe SLS, pour son accueil au sein de son groupe ainsi que pour m'avoir encadré durant les trois années de cette thèse.

Je tiens tout particulièrement à remercier Monsieur Frédéric Rousseau, maître de conférences à l'université Joseph Fourier de Grenoble, pour son encadrement, ses conseils toujours pertinents, sa confiance et sa grande capacité d'écoute. Je ne disposerai jamais d'assez de place ni des mots justes pour lui exprimer mon respect et ma gratitude.

Je remercie Monsieur Pierre Gentil, directeur de l'école doctorale EEATS, de m'avoir fait l'honneur de présider le jury de ma thèse. Que messieurs Bruno Rouzeyre de l'université de Montpellier et Jean Luc Philippe de l'université de Bretagne sud reçoivent ici l'expression de ma gratitude tant pour leur participation au jury en tant que rapporteurs que pour leurs conseils en vu de l'amélioration du manuscrit.

Je remercie également Monsieur Bernard Courtois, directeur du laboratoire TIMA, de m'avoir accueilli au sein de son laboratoire.

Enfin, j'aimerais aussi remercier pour leur sympathie tous les membres du groupe SLS que j'ai eu le plaisir de côtoyer. Merci notamment à Sonja pour sa gaieté et sa bonne humeur contagieuse, à Nacer pour ses conseils et les discussions que nous avons pu avoir, à Wander pour m'avoir initié aux méandres du flot ROSES.

Je suis aussi reconnaissant aux thésards (Ivan, Arif, Marcio, Yanick, Lisane, Ferid, Youssef, Youngchul) qui ont partagé mon bureau, mes repas et pauses cafés pour les bons moments passés et pour nos discussions aussi bien professionnelles que non professionnelles. Ivan, j'espère ne pas avoir trop "exploiter ton français". Arif, je suis content de t'avoir rencontré et je te souhaite une bonne continuation dans ta carrière de PDG. Enfin, j'aimerai remercier Yanick pour sa confiance en mon potentiel.

Merci à tous les thésards de l'équipe SLS (Fred, Adriano, Wassim, Lobna, Aimen, Benaoumeur, Katy, Marius, Anouar, Gabriela …) et aux post-doc (Xi et Patricia) avec lesquels j'ai eu le plaisir de travailler et avec qui j'ai pu partagé quelques moments de détente durant ces trois ans. Je remercie Damien pour m'avoir accueilli dans le groupe et transmis ses connaissances.

Un dernier remerciement à ma mère pour son soutien et ses encouragements, et à mon frère pour m'avoir indiquer les obstacles à éviter au cours du chemin tortueux qui mène à une thèse.

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

Introduction à la conception des systèmes monopuces

Sommaire

1.1 Introduction

L'invention du transistor et son intégration en grand nombre dans un même composant a révolutionné les possibilités de calcul des systèmes informatiques et a permis leur miniaturisation. Une révolution s'est depuis opérée et on assiste maintenant à une course en avant dans les capacités d'intégration et les fréquences de fonctionnement des systèmes numériques synchrones. On estime que la contribution apportée par l'augmentation du nombre de transistors à l'augmentation des performances des microprocesseurs a été plus grande d'un ordre de magnitude que la contribution apportée par l'augmentation des fréquences d'horloge au cours des deux dernières décennies [Cul99a].

Avec les technologies de fabrication actuelles, certains circuits intégrés contiennent plusieurs centaines de millions de transistors. L'évolution de la technologie permet la réalisation de circuits de plus en plus complexes grâce à de plus grandes capacités d'intégration. Il est aujourd'hui possible d'intégrer dans une même puce pratiquement tous les composants nécessaires au bon fonctionnement d'un système électronique, et qui étaient auparavant intégrés sur une carte. On parle alors de systèmes monopuces. On utilise couramment le terme SoC dérivé du nom anglais "system-on-chip". Aujourd'hui, une grande partie des circuits intégrés qui sont développés sont des systèmes monopuces. De tels systèmes sont composés de processeurs (à usage général ou dédiés) et de composants matériels spécifiques, interconnectés par des bus de communications.

Ils se sont aujourd'hui diffusés dans l'industrie électronique et on les retrouve dans diverses applications telles l'automobile, la téléphonie mobile, les consoles de jeu, la photo numérique ou la télévision numérique … Ils visent donc essentiellement des applications embarquées requérant des performances élevées, mais leur coût les restreint à des applications destinées à un marché de masse.

Les systèmes monopuces d'aujourd'hui qui intègrent des millions de transistors sont difficilement comparables aux premiers circuits intégrés composés de quelques transistors. Ils offrent d'ailleurs des capacités de calcul bien plus importantes que les ordinateurs de l'époque. Les circuits VLSI ("*Very Large Scale Integrated*") représentaient un changement dans la taille des circuits et le nombre de transistors. Les systèmes monopuces poursuivent ce changement d'échelle, mais le principal changement concerne l'intégration de processeurs logiciels. La conception des systèmes monopuces requiert maintenant la maîtrise des métiers du génie logiciel. Pour s'adapter à cette évolution, les concepteurs remettent en cause leur façon d'aborder la conception des circuits intégrés. L'introduction de nouvelles méthodologies de conception, avec les outils appropriés, est indispensable pour maîtriser la complexité croissante des circuits et limiter les temps de conception. Ces temps croissent de façon continue alors que les temps de mise sur le marché des composants électroniques sont aujourd'hui de plus en plus courts.

Ce chapitre présente les problèmes rencontrés dans la conception des systèmes monopuces. Les systèmes monopuces y sont brièvement décrits et analysés, puis la problématique de cette thèse est introduite, ainsi que ses objectifs et les contributions apportées.

1.2 Contexte : l'évolution des circuits intégrés numériques

1.2.1 Qu'est-ce qu'un système monopuce ?

Avant d'aborder la problématique visée par cette thèse, il est nécessaire de définir ce qu'est un système monopuce. Un système monopuce est un système composé d'un ensemble de processeurs, de DSP, de mémoires, de coprocesseurs et de périphériques matériels regroupés autour d'un réseau de communication et généralement intégrés sur un même substrat. Dans le cas d'un système intégré dans un même composant mais sur différents substrats, on parlera plutôt de système "encapsulé" (en anglais "System-on-Package" ou SoP). Le réseau de communication est souvent composé d'un ensemble de bus partagés, interconnectés par des adaptateurs de bus (aussi appelé pont, de l'anglais "bridge").

Par exemple, la figure 1.1 montre l'architecture du système monopuce Nomadik [STM04a, STM04b] développé par la société STMicroelectronics. Ce système est destiné principalement au marché de la téléphonie mobile mais peut aussi servir pour la réalisation d'applications multimédias pour des systèmes embarqués tels que les PDA ("Personal Digital Assistants").

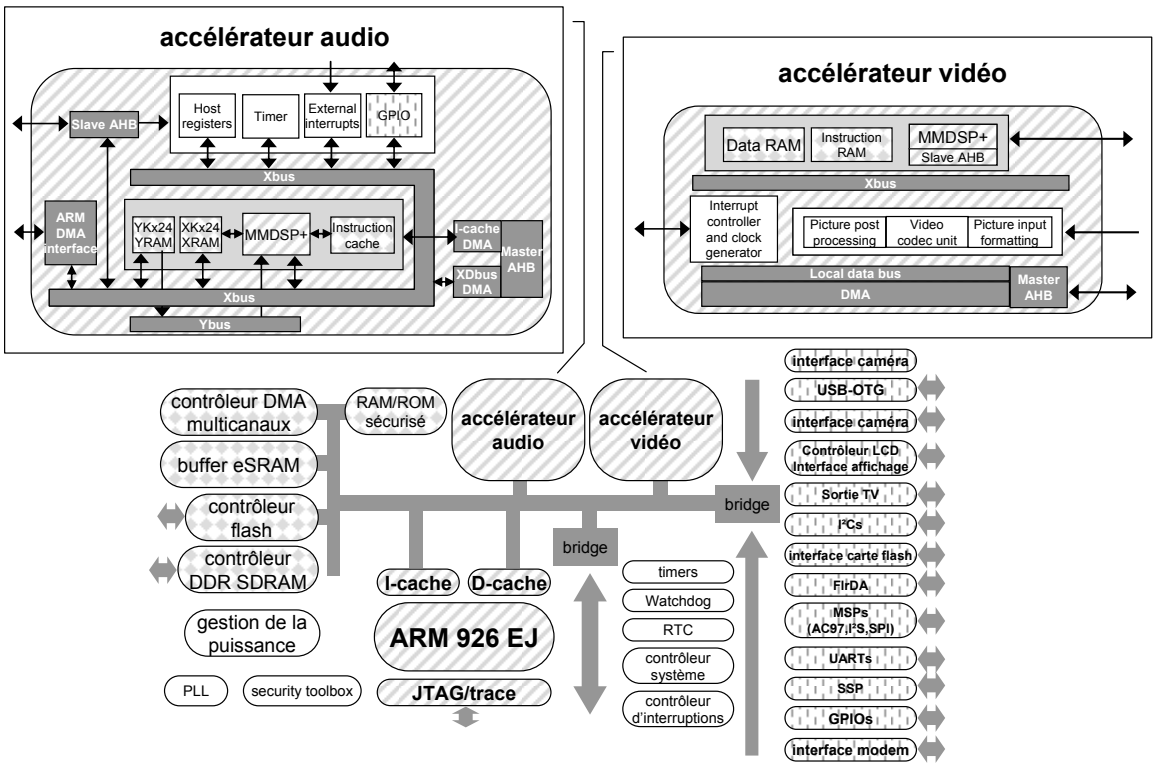


Figure 1.1 - L'architecture du système Nomadik (adapté de [STM04a])

Le système Nomadik est construit autour d'un processeur RISC ARM926 EJ [ARM04a]. Autour de ce processeur principal se greffent deux sous-systèmes processeurs en charge du traitement de l'image et du traitement audio. Ils déchargent le processeur principal des opérations de calcul, qui n'a alors plus que pour rôle de contrôler les différentes opérations. Le sous-système de traitement audio est basé sur un DSP VLIW adjoint de mémoires caches et de périphériques de contrôle. L'accélérateur vidéo sert pour l'encodage ou le décodage vidéo au format MPEG en temps réel. Ce sous-système est

également basé sur un DSP VLIW (MMDSP+ pour “MultiMedia-DSP-Plus”). Des coprocesseurs permettent d’accélérer les opérations de calcul afin de respecter les contraintes de temps réel : *video codec unit*, *picture post processing* et *picture input formatting*. La réalisation du traitement audio et vidéo en logiciel permet de supporter une multitude de formats et standards avec les mêmes ressources matérielles.

Le système possède de la mémoire embarquée de type SRAM et ROM avec un accès sécurisé. Il possède également deux contrôleurs mémoires permettant d’accéder à des mémoires externes SDRAM et/ou flash. La mémoire RAM embarquée sert de mémoire cache permettant ainsi de diminuer la latence mémoire, la puissance consommée par l’accès aux mémoires externes et la bande passante interne nécessaire, tout spécialement pendant l’encodage vidéo. L’interfaçage du système avec les autres composants de l’application est réalisé par les différents périphériques d’entrée-sortie. On peut ainsi interfacer le système aussi bien avec un capteur CCD qu’un contrôleur LCD, une liaison USB, des cartes mémoires flash (MMC/SD/MS), une liaison série (UARTs) … Le réseau de communication est un “crossbar AMBA” [ARM04b].

Il est possible de schématiser un système monopuce suivant le modèle de la figure 1.2. Un système peut être modélisé comme un ensemble de composants de base connectés autour d’un réseau de communication. Parmi ces composants, on trouve des sous-systèmes processeurs, des coprocesseurs matériels, des périphériques d’entrée-sortie et des mémoires. Un sous-système processeur est un processeur éventuellement accompagné d’un ensemble de composants de base nécessaire à son fonctionnement (mémoire ROM/RAM, contrôleur mémoire, contrôleur d'interruptions, …). Un système monopuce intègre de la mémoire nécessaire au fonctionnement d’un processeur, ou de la mémoire partagée pour la communication entre plusieurs composants ou de la mémoire tampon pour cacher la latence du système et amortir les variations des flux de données. Il comprend également des coprocesseurs matériels pour le contrôle du système, ou pour des opérations de calcul. La quatrième catégorie de composants comprend les périphériques d’entrée-sortie, qui assurent la liaison entre le système et l’extérieur.

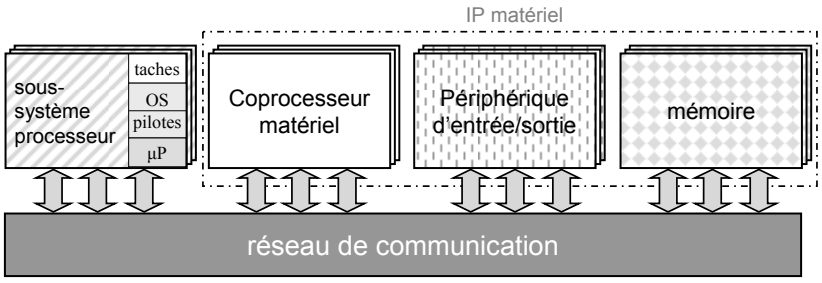


Figure 1.2 - Modèle générique d’un système monopuce

Le système Nomadik qui vient d’être présenté rentre parfaitement dans ce cadre conceptuel, qui servira de point de départ à la définition de la problématique visée par cette thèse. Il permet aussi de se rendre compte qu’un sous-système processeur peut représenter un cœur de processeur avec de la

mémoire cache et une interface de déboguage (ARM926EJ), mais peut aussi être plus complexe. Le sous-système de traitement vidéo pourrait tout aussi bien être considéré comme un système monopuce s'il était analysé indépendamment. Il rentre ainsi tout à fait dans le schéma conceptuel de la figure 1.2.

Le système Nomadik a été conçu de façon à faciliter le développement d'applications afin de diminuer les temps de mise sur le marché des produits finaux. En effet, la conception du logiciel embarqué prend une place croissante dans le processus de développement des systèmes monopuces. Il existe deux raisons à cette évolution. La réalisation d'applications plus évoluées en terme de performances mais aussi de fonctionnalités disponibles, implique du logiciel embarqué plus complexe mais aussi plus important en taille. La deuxième raison est liée à l'utilisation de circuits matériels de plus en plus complexes pour supporter ces nouvelles applications, et donc plus difficiles à programmer par le concepteur du logiciel. Le système Nomadik offre deux solutions pour diminuer le temps de développement du logiciel. La première est apportée par le processeur ARM926EJ qui offre une accélération matérielle du langage Java permettant la réalisation de l'application avec ce langage de haut niveau sans sacrifier les performances. La deuxième solution est apportée par la définition d'une structure logicielle en couche (figure 1.3) et d'API ("Application Program Interface"). Une API est une interface de programmation d'un composant logiciel, qui se présente généralement sous la forme d'un ensemble de fonctions.

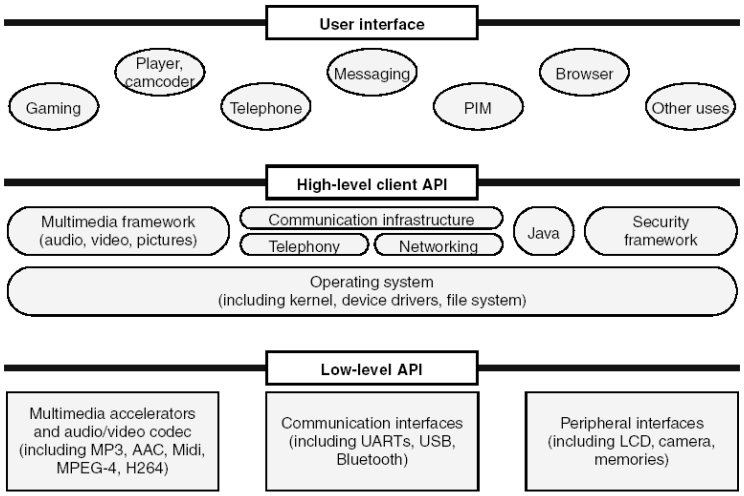


Figure 1.3 - L'architecture logicielle du système Nomadik (extrait de [STM04a])

Cette architecture logicielle a pour objectif d'améliorer la portabilité des applications en définissant un standard d'interfaces. La couche inférieure sert à l'abstraction physique pour découpler l'application de la plateforme sous-jacente. Elle est basée sur un standard d'API pour les périphériques. Le portage d'une application à une autre plateforme respectant le standard d'interface est donc aisé, ce qui autorise des évolutions rapides de la plateforme. La portabilité des applications sur les produits supportés permet également de réduire les efforts de développement du logiciel et de diminuer les temps de mise sur le marché des produits finaux.

1.2.2 Pourquoi conçoit-on et utilise-t-on des systèmes monopuces ?

Afin de répondre à la demande toujours croissante de nouvelles fonctionnalités pour des applications telles que les applications multimédia, les systèmes monopuces sont une des solutions pour maîtriser la complexité de ces applications et pour respecter les contraintes sévères de temps de mise sur le marché.

L'utilisation de processeurs permet d'avoir des circuits programmables, qui sont évolutifs et flexibles. La conception d'un système monopuce peut alors commencer avant la finalisation d'une norme ou avant la définition finale du produit. Il est envisageable de faire évoluer le système tout au long de sa conception car la modification du logiciel est beaucoup plus facile et représente un effort moindre que le changement du matériel. On peut ainsi réduire les temps de mise sur le marché des circuits intégrés en amorçant le processus de conception dès que les normes sont à peu près définies. Le système peut évoluer non seulement pendant les étapes de sa conception mais aussi tout au long de la vie du produit. Il est ainsi possible de le faire évoluer avec l'évolution d'une norme, ajouter de nouvelles fonctionnalités ou le support de nouvelles normes, ou bien tout simplement améliorer la qualité du produit avec par exemple le perfectionnement d'algorithmes de traitement d'images. Cette évolution est beaucoup plus rapide qu'elle ne le serait s'il fallait modifier le circuit. Cette flexibilité permet également la différenciation du produit final. En effet, l'utilisation de processeurs laisse une plus grande liberté d'utilisation du circuit qu'un ASIC. Le concepteur de l'application peut ainsi adapter le produit à son application afin de le différencier d'un produit concurrent utilisant le même circuit. La différenciation s'effectue à la fois en terme de fonctionnalités disponibles mais aussi de qualité du produit (traitement d'images par exemple).

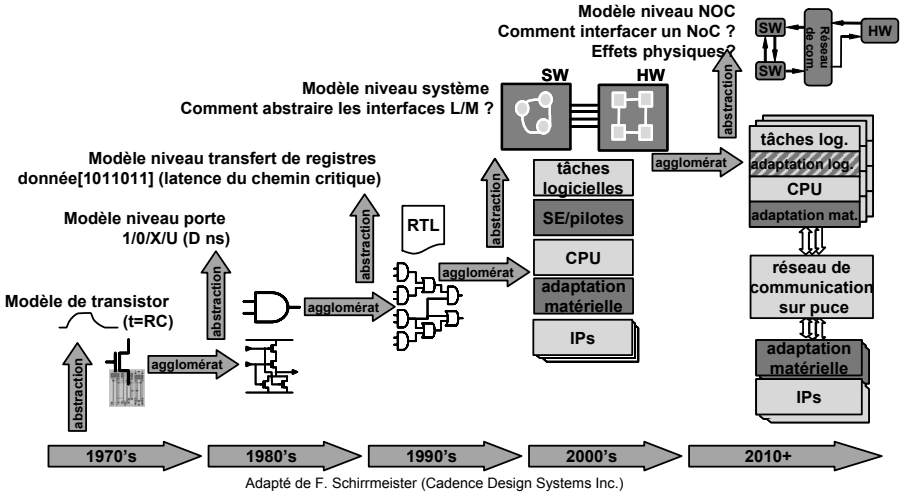
Un circuit programmable possède également l'avantage de pouvoir être utilisé pour différentes applications. Cela permet de toucher un marché plus grand, et de mieux rentabiliser le circuit grâce à des ventes plus importantes. Les coûts de conception et de fabrication de tels circuits allant croissant pour des raisons qui sont évoquées plus loin, ceci n'est pas négligeable. Le système Nomadik que nous avons présenté précédemment peut ainsi être utilisé pour différentes applications multimédia et n'est pas limité à la téléphonie mobile.

Si la réalisation d'applications en logiciel a l'avantage de la flexibilité, cette solution se heurte néanmoins au traditionnel problème du logiciel. L'utilisation de composants matériels spécifiques, ou ASIC ("Application Specific Integrated Circuit"), offre de meilleures performances pour une puissance plus faible. Une approche purement matérielle ou purement logicielle n'est pas toujours bien adaptée à certaines applications. Certaines applications ne peuvent pas être réalisées entièrement en logiciel, mais une réalisation complètement matérielle n'est pas totalement satisfaisante car pas assez flexible. Traditionnellement, ces circuits étaient jusqu'alors réalisés avec une approche hybride en intégrant des processeurs et des ASIC sur une carte. Aujourd'hui, la technologie permet d'intégrer ces composants dans une même puce afin de miniaturiser les systèmes. C'est ce qui a amené au

développement des systèmes monopuces, assemblage de processeurs pour leur flexibilité et de blocs matériels pour leurs performances.

1.2.3 Influence de l'évolution des technologies de fabrication

Les capacités d'intégration des technologies de fabrication autorisent le développement d'applications de complexité croissante. Cela répond à une double demande d'avoir des circuits intégrés plus performants et offrant plus de fonctionnalités pour une surface moindre (miniaturisation). Même si l'élément de base à intégrer est resté le même, c'est-à-dire le transistor, pour le concepteur il ne s'agit plus tellement d'intégrer des transistors mais des processeurs, mémoires ou bus partagés sur une même puce. Les circuits actuels allant jusqu'à plusieurs centaines de millions de transistors, concevoir ces circuits en décrivant manuellement l'assemblage de chaque transistor représenterait une masse de travail excessive.



Adapté de F. Schirrmeister (Cadence Design Systems Inc.)

Figure 1.4 - De l'ASIC au SoC [Jer04]

Pour gérer la complexité des circuits intégrés traditionnellement évaluée en nombre de transistors, les méthodologies de conception ont toujours reposé sur trois principes de bases : la décomposition du problème, la réutilisation de composants et l'abstraction. La décomposition du circuit en plusieurs parties plus simples à concevoir permet de simplifier un problème trop compliqué à maîtriser en le décomposant en plusieurs sous-problèmes que l'on sait gérer. La réutilisation de composants pour concevoir ces différentes parties permet d'accélérer la conception des circuits. Pour maîtriser la complexité des circuits, la nature des composants réutilisés a continuellement évolué, du transistor au processeur en passant par des portes logiques et des registres. Cette évolution a pu se faire grâce à l'utilisation de modèles de spécification du circuit avec des niveaux d'abstraction croissants. Si les caractéristiques électriques des transistors sont nécessaires pour les assembler, l'étude analogique d'un circuit de plusieurs millions de transistors serait ingérable. Il est donc nécessaire d'abstraire les problèmes pour se concentrer sur les problèmes significatifs. Comme le montre la figure 1.4, on a commencé par abstraire les zones de dopage et les masques, pour concevoir le circuit comme un

assemblage de transistors, puis comme un assemblage de portes logiques et finalement comme de la logique combinatoire et des registres avec la synthèse logique.

Les méthodes de conception ont donc continuellement évolué pour pouvoir gérer la complexité croissante des circuits intégrés. L'abstraction permet de concevoir des circuits plus rapidement, ouvrant la voie à l'intégration d'un plus grand nombre de composants, jusqu'à ce qu'on soit limité par de nouveaux problèmes. On assiste aujourd'hui à un problème de productivité des concepteurs, et à un accroissement des temps de conception. Face à la complexité des systèmes monopuces, les méthodes de conception existantes et l'expérience du concepteur sont insuffisantes pour respecter des temps de mise sur le marché extrêmement courts. Pour comprendre comment concevoir des systèmes monopuces de plus en plus complexe dans des délais raisonnables en gardant le contrôle sur le résultat final, il nous faut comprendre les difficultés rencontrées pour leur conception afin de passer à un niveau d'abstraction supérieur.

1.3 Problématique : la nécessité de nouvelles méthodes de conception

1.3.1 Contraintes physiques et limites des méthodes classiques de conception

a) Limites physiques

L'augmentation des performances des circuits intégrés est liée à la finesse de gravure des circuits. Sa diminution a permis d'intégrer un nombre croissant de transistors sur une même surface de silicium, tout en permettant d'augmenter les fréquences de fonctionnement. Avec les finesses de gravure atteintes par les nouvelles technologies, un certain nombre de limites physiques pose des problèmes en terme de fiabilité et de puissance consommée [Bry01]. Les interférences électromagnétiques existant entre deux lignes proches, dues à leur couplage capacitif ou à leur inductance mutuelle, sont des sources de bruit. Avec l'augmentation en fréquence des circuits et la plus grande finesse de gravure, elles ne sont plus négligeables. De même, le circuit est plus sensible aux rayons cosmiques qui risquent de charger ou décharger inopinément une capacité ("soft errors" en anglais).

Une nouvelle limitation physique va également apparaître dans les années à venir. Avec l'augmentation en fréquence des circuits, les temps de propagation des signaux ne vont plus être négligeables. On découpe aujourd'hui le circuit en domaines d'horloges pour décrire les parties du circuit sur laquelle l'hypothèse de synchronisme est respectée. Les communications globales entre des composants du système appartenant à des domaines d'horloges différentes sont par contre asynchrones. On parle alors de circuits Globalement Asynchrone, Localement Synchrone (GALS). D'autant plus que l'on peut avoir des composants fonctionnant à des fréquences différentes sur la même puce, soit parce qu'un composant ne peut atteindre la fréquence des autres composants, soit pour diminuer la puissance dissipée en fonction des performances souhaitées. Le découpage du circuit

en domaines d'horloges est donc lié soit à l'utilisation d'horloges de fréquences différentes, soit au non respect de l'hypothèse de synchronisme à cause des temps de propagation sur les lignes utilisées pour les communications globales.

Une autre limitation liée aux interconnexions concerne la place occupée par celles-ci. Elle devient de moins en moins négligeable par rapport à la taille des transistors, et nécessite de multiplier les niveaux de métal sous peine d'agrandir la surface du circuit. Cette solution a un coût et n'est pas extensible à l'infini.

Les nouvelles possibilités de la technologie et les effets physiques qui vont en découler, orientent la conception dans le même sens : "le besoin de prendre en compte les communications". Considérer que la communication se limite aux interconnexions n'est plus possible. Cette tendance a conduit à l'utilisation de bus de communication de plus en plus compliqués, et les réseaux de communication vont probablement devenir encore plus compliqués comme nous le verrons dans le chapitre 2. La communication est alors beaucoup moins prévisible qu'avec l'utilisation uniquement d'interconnexions dédiées et doit donc être traitée différemment.

Les différents composants de calcul deviennent eux aussi de plus en plus compliqués. La puissance consommée augmente de plus en plus. La solution technologique consistant à réduire la tension d'alimentation pour une plus faible consommation entraîne une plus faible immunité au bruit. La consommation doit donc être prise en compte lors de la conception du système, en utilisant par exemple les modes basses consommations de certains processeurs.

Les limites physiques influent donc sur la conception du système. Elles ne se limitent pas à un problème technologique mais doivent être prise en compte dès les premières étapes de la conception du système.

b) Problème de productivité

Les systèmes monopuces sont des systèmes multiprocesseurs hétérogènes. Les flots de développement logiciel et matériel sont généralement séparés. Le logiciel est conçu une fois le matériel terminé, ce qui allonge les délais. Ce problème s'accentue dans la mesure où les systèmes en se complexifiant deviennent plus difficiles à programmer. Cela pose également des problèmes pour prévoir si le système final est capable de fournir les performances attendues car on ne peut évaluer ses performances qu'une fois le matériel conçu. Compte tenu du coût de conception d'un tel système, il n'est pas acceptable d'attendre que le système soit entièrement conçu pour s'apercevoir que le système ne respecte pas la spécification. Dans la mesure où le résultat est incertain, il n'est également pas possible d'optimiser le système, ce qui est un frein pour améliorer la qualité des systèmes.

Les méthodes de conception n'ayant pas radicalement évolué depuis l'apparition de la synthèse logique, les temps de conception sont devenus excessifs. On parle maintenant de "lacune de productivité" (*design productivity gap* en anglais) pour se référer à l'écart existant entre l'évolution du nombre de transistors que l'on peut mettre sur une puce et le nombre moyen de transistors d'un circuit

qu'un ingénieur peut concevoir en une journée de travail. Pour contrebalancer ce manque de productivité, les équipes de conception doivent être très grandes, ce qui implique un coût très important. De plus, le coût des masques devient d'autant plus élevé que l'on diminue la finesse de gravure des transistors.

Ces systèmes sont donc complexes et sont soumis à de fortes contraintes (performance, coût, délai de réalisation, …). La conception de tels systèmes nécessite de faire des compromis entre les différentes contraintes. Augmenter la fréquence de fonctionnement d'un processeur permet par exemple d'augmenter les performances au détriment de la puissance consommée. Ces différents choix sont guidés par l'expérience du concepteur qui essaie de tenir compte des différents paramètres afin de respecter les différentes contraintes de conception. Les systèmes ont aujourd'hui atteint une telle complexité qu'il devient humainement difficile de tenir compte de tous les paramètres et de leurs multiples interactions. Il est donc nécessaire d'établir une méthodologie systématique permettant au concepteur de maîtriser la complexité des SoC, afin de réduire les temps et les coûts de conception tout en améliorant la qualité des systèmes [Mag02].

A cette fin, les principes de base de la conception des circuits intégrés doivent être redéfinis en fonction des problèmes posés par la conception des systèmes monopuces.

1.3.2 Principes de base de la conception des systèmes monopuces

a) Découplage calcul/communication

La communication entre différents composants hétérogènes est une des difficultés de la conception des systèmes monopuces. Les communications dans les systèmes monopuces se complexifient et sont d'une importance décisive pour les performances du système. Le problème de productivité des concepteurs requiert un nouveau paradigme tenant compte des problèmes physiques posés par la communication dans les systèmes monopuces.

Le découplage du calcul et de la communication sépare les problèmes afin de mieux maîtriser cette complexité. Pour le concepteur de l'application, il est important de savoir comment envoyer des données d'une tâche à une autre ou d'une tâche à un coprocesseur matériel. Il est nécessaire de connaître les fonctions à appeler pour envoyer une donnée, mais il est inutile de connaître comment la communication se fait. Les composants de calcul sont conçus avec l'hypothèse que le réseau de communication fournit les performances requises. Cela facilite leur conception mais aussi la conception du système qui consiste alors à connecter ces composants à un réseau de communication. Du point de vue du concepteur du réseau, les informations importantes sont les besoins en bande passante/latence d'un composant. Le type et la signification des données transportées ne sont pas d'une grande importance pour lui. Les ressources de communication peuvent donc être conçues indépendamment des ressources de calcul.

On parle alors de conception basée sur les interfaces (“interface-based design” en anglais), car on ne s’intéresse plus à la manière dont est réalisé le composant, mais seulement à son interface pour savoir s’il peut-être connecté au réseau de communication [Row97]. La conception de systèmes monopuces consiste alors à assembler des composants autour d’un réseau de communication.

b) Réutilisation

Pour réduire les coûts et les temps de conception, un concept important dans les systèmes monopuces est celui de la réutilisation des composants. Des modèles de composants matériels (au niveau masque ou transfert de registres/RTL) sont considérés comme de la “propriété intellectuelle”, ou IP (“Intellectual Property” en anglais). On parle donc d’IP pour se référer à un composant ou bloc matériel réutilisable. Les processeurs sont également réutilisables. L’utilisation de processeurs et d’IP permet de réduire le temps de conception des SoC. Un système monopuce peut donc être représenté par le schéma de la figure 1.5.

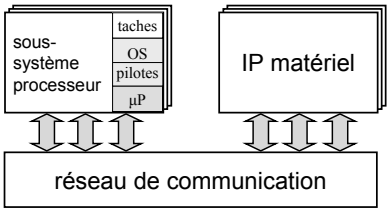


Figure 1.5 - Système monopuce : un réseau de composants réutilisables

Si l’on conçoit les systèmes monopuces à partir de composants réutilisables, le développement des systèmes consiste alors à assembler les composants pour les intégrer dans un même circuit. Le système peut être vu comme un ensemble de noeuds de calcul logiciel (processeur) ou matériel (IP) communiquant à travers un réseau de communication. La conception des SoC consiste alors à choisir les différents nœuds de calcul et à les faire communiquer ensemble. Les composants de communication peuvent aussi être déjà conçus et réutilisés après configuration.

c) Abstraction

Concevoir des systèmes monopuces au niveau transfert de registres reste difficile même si l’on utilise des composants réutilisables. Il est aujourd’hui nécessaire d’élever le niveau d’abstraction des flots de conception classiques pour maîtriser la complexité des SoC. Dans les premières étapes de la conception, l'abstraction dispense le concepteur de la gestion des problèmes liés à l'hétérogénéité du système et aux limites physiques des interconnexions. Pour cela, des modèles de haut niveau ont été proposés pour abstraire les communications dans systèmes monopuces [Ros04].

Pour aborder la conception du système dans une seule et même approche cohérente, les composants aussi bien logiciels que matériels sont modélisés avec un modèle unique. Des flots de conception séparés du logiciel et du matériel ne permettent ni de respecter les délais de conception, ni de garantir que le système répondra aux exigences avant son prototypage. La conception conjointe du

matériel et du logiciel est un défi pour la réalisation d'un flot de conception système, préambule à l'obtention de circuits de meilleure qualité en un temps plus court.

1.3.3 Flot de conception des systèmes monopuces

Les principes qui viennent d'être évoqués ont servi de base pour la définition d'un flot de conception au niveau système [Jer02, Döm98].

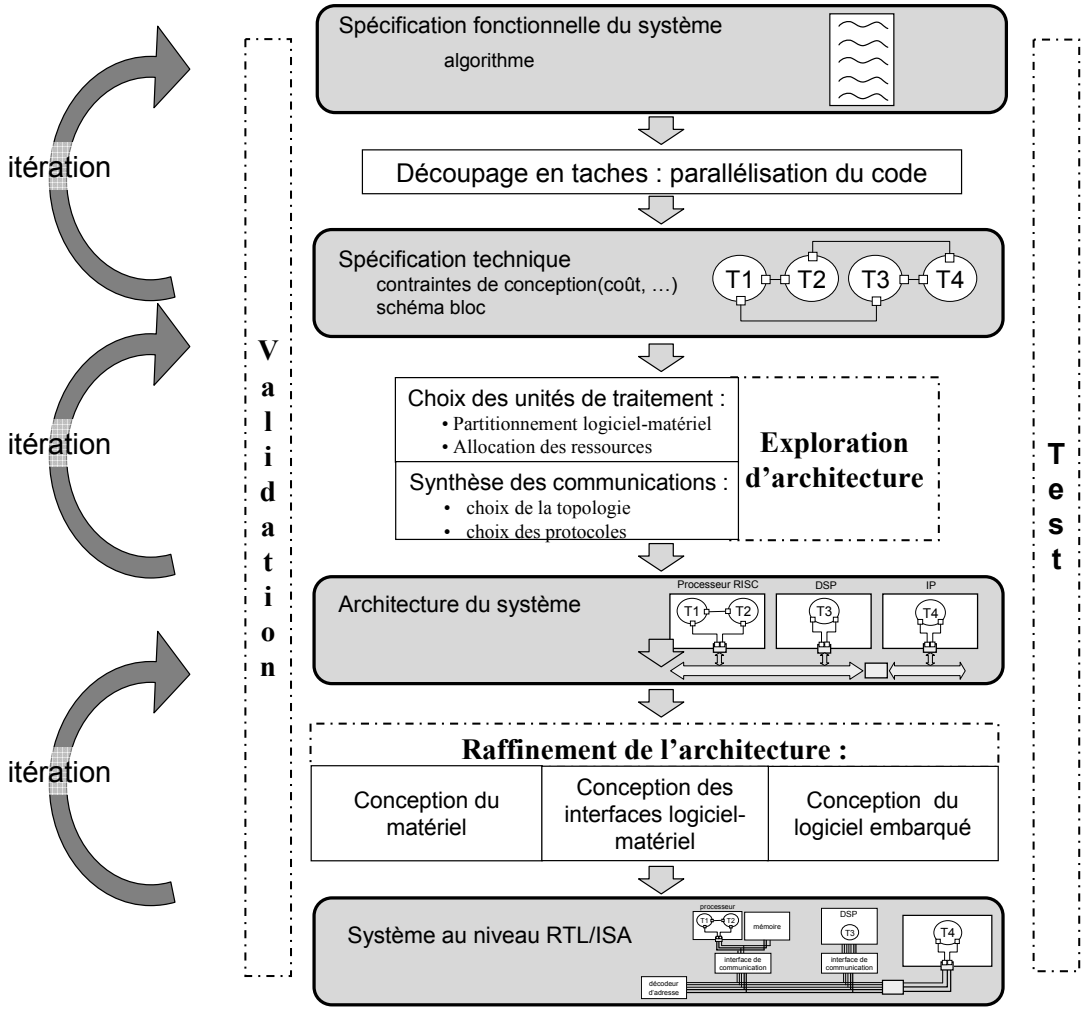


Figure 1.6 - Flot de conception des systèmes monopuces

La figure 1.6 montre ce flot tel qu'il est généralement présenté. Ce flot part d'une spécification fonctionnelle du système, qui peut par exemple être décrit sous forme d'un algorithme. Cet algorithme est ensuite découpé en tâches. On a alors la spécification technique du système que l'on modélise par un ensemble de tâches communicantes. On cherche ensuite à fixer l'architecture du système en évaluant les différents choix possibles. L'utilisation d'une architecture adaptée à une application spécifique permet de mieux répondre aux besoins de cette dernière. L'exploration d'architecture se fait en deux étapes. On choisit tout d'abord si l'on réalise les différentes tâches en logiciel ou matériel afin d'offrir la flexibilité attendue tout en offrant les performances nécessaires puis les éléments (CPU, mémoire) sont choisis et taillés sur mesure en vue d'obtenir les meilleurs coûts et performances. La

deuxième étape de l'exploration d'architecture est la synthèse des communications qui consiste à fixer la topologie du réseau et le protocole de communication. Le raffinement de l'architecture consiste à intégrer dans un même circuit les différents composants du système. Les IP et les processeurs doivent être remplacés par leurs modèles RTL et le logiciel de bas niveau est conçu (pilotes de communication et système d'exploitation).

Comme il n'existe pas de méthodes systématiques, chaque étape suit un processus itératif. Si le système conçu à l'issu des différentes étapes ne satisfait pas aux contraintes initiales, il est nécessaire de revenir en arrière dans le processus de conception.

1.3.4 Nécessité des interfaces de communication

a) Nécessité des interfaces de communication

Les systèmes monopuces sont conçus en assemblant des composants réutilisables autour d'un réseau de communication. On applique ainsi les deux principes évoqués précédemment : la réutilisation et le découplage du calcul et la communication. L'inconvénient de cette méthodologie est que les problèmes d'interaction entre le calcul et la communication réapparaissent quand on veut assembler les composants du système au niveau RTL. Découpler le calcul et la communication permet de concevoir les composants de calcul et de communication indépendamment et de pouvoir les réutiliser facilement. Comme nous l'avons vu précédemment, les composants de calcul et le réseau de communication sont généralement hétérogènes en termes de protocoles de communication, bande passante/latence, interface physique (taille de bus, bande passante, signaux de contrôle), domaines d'horloges. Mais ces composants ayant été conçus indépendamment, ils n'ont pas été prévus pour fonctionner ensemble et on a alors des composants incompatibles. Un problème de conception des systèmes monopuces est l'assemblage de ces composants hétérogènes et leur communication. Cet assemblage implique l'utilisation de composants d'interfaçage pour connecter les composants de calcul au réseau de communication (figure 1.7). Ils servent d'adaptateurs entre les composants de calcul et le réseau de communication. On parle d'interfaces de communication pour qualifier ces adaptateurs.

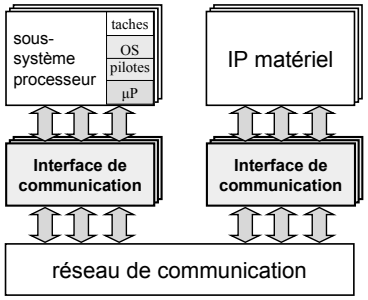


Figure 1.7 - L'utilisation d'interfaces de communication

L'utilisation d'adaptateurs découple le calcul et la communication, permettant ainsi la réutilisation des différents composants. En effet, le concepteur peut assembler des composants sans savoir s'ils sont

compatibles. Le changement d'un composant ou du réseau nécessite de concevoir un nouvel adaptateur mais permet de garder indemne une partie du système. Le problème devient alors la conception des interfaces, qui peut prendre du temps.

b) Difficultés de leur conception

La conception des interfaces de communication est une tâche laborieuse, et surtout source d'erreurs. Cette difficulté est notamment due à l'étendue des connaissances que requiert leur conception : bonne connaissance des composants, protocoles de communication, intégration logiciel/matériel. Elle implique en effet plusieurs composants (processeurs, pilotes de communication, IP et réseau). Par exemple, pour concevoir une interface de communication entre un processeur MIPS et un bus AMBA, le concepteur doit être familier avec la spécification de ces deux composants.

De façon à concevoir simultanément l'interface de communication et le logiciel, les concepteurs logiciels et matériels doivent s'accorder sur la façon dont le logiciel accède à l'interface (plan mémoire, utilisation d'interruptions …). De plus, cela nécessite de la part du concepteur de connaître le protocole de communication. Par exemple, une communication par mémoire partagée suppose de connaître les mémoires locales et globales du système et la manière dont l'interface peut y accéder. La diversité des connaissances requises et le nombre de paramètres à gérer rendent la conception laborieuse et source d'erreurs. Cette connaissance est d'autant plus difficile à acquérir qu'il existe de nombreux protocoles, une grande diversité de composants, et plusieurs modèles de communications (mémoire partagée/passage des messages, point à point/collective) …

De plus, l'hétérogénéité du système se manifeste au niveau de l'interface, ce qui rend leur conception encore plus ardue. Les SoC sont composés de composants hétérogènes en terme de :

- ⇨ fonctionnalité : calcul ou communication
- ⇨ protocoles
- ⇨ nature : matérielle ou logicielle
- ⇨ domaines d'horloges : système Globalement Asynchrone, Localement Synchrone
- ⇨ interfaces physiques

La difficulté de conception des interfaces est problématique dans la mesure où la grande variété de composants existants limite la réutilisation des interfaces de communication. Modifier un composant du système implique de concevoir à nouveau une interface de communication. Réutiliser des interfaces de communication requiert d'imposer des restrictions sur le choix des composants du système pour pouvoir se satisfaire d'un nombre limité d'interfaces de communication réutilisables. On risque alors d'avoir des systèmes non optimisés et des interfaces mal adaptées au système. Le temps de conception des interfaces de communication n'est donc pas négligeable. Les travaux menés durant cette thèse visent à accélérer leur conception.

1.4 Motivations et objectifs

1.4.1 Vers un flot de synthèse au niveau système

Le travail de cette thèse s'inscrit dans l'optique de la réalisation d'un flot de synthèse au niveau système. Comme nous l'avons vu précédemment, le processus de conception d'un système monopuce est itératif. Ce processus itératif implique d'évaluer différents choix de conception et de revenir en arrière dans le processus si le résultat ne satisfait pas les contraintes imposées. Plus on va descendre dans le processus de conception, plus l'espace des solutions agrandit comme l'illustre la figure 1.8.

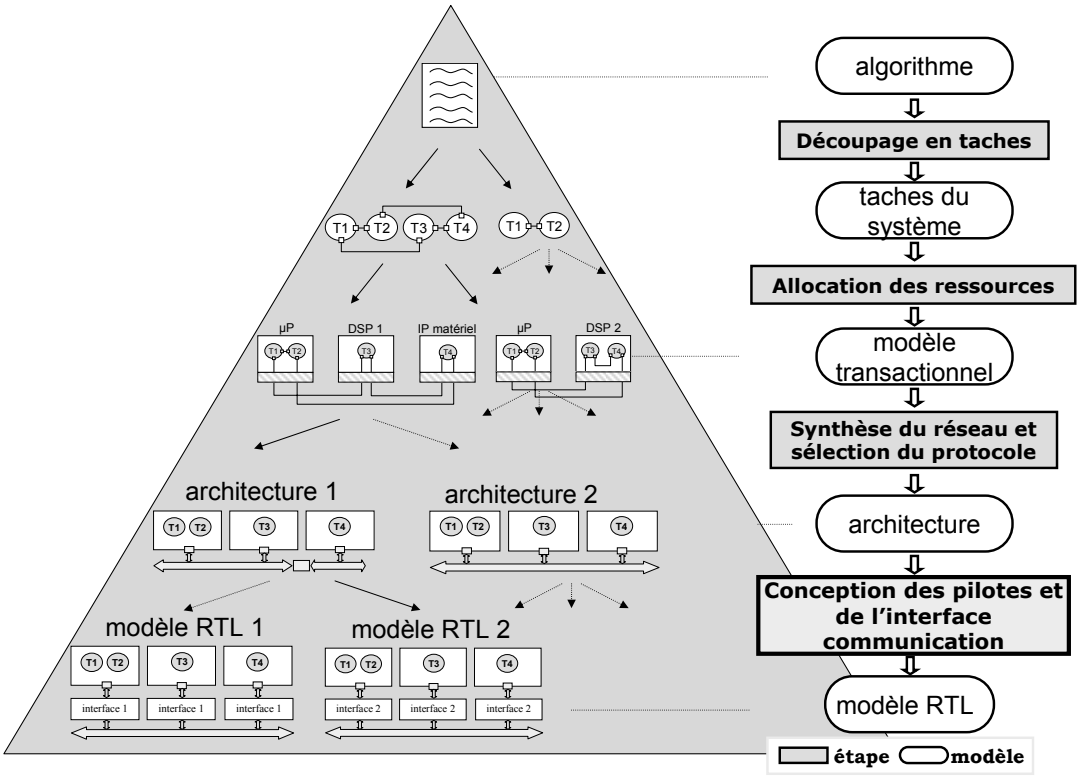


Figure 1.8 - L'exploration des choix de conception

Comme l'exploration manuelle de l'ensemble des solutions est inenvisageable compte tenu du nombre de solutions possibles et du temps nécessaire pour obtenir un modèle RTL du système à partir de la spécification fonctionnelle, celle-ci doit être automatisée. L'objectif de la synthèse au niveau système est de faire communiquer des tâches logicielles et des IP matériels entre eux, tout comme la synthèse logique connectait des portes logiques, en explorant les différents choix de conception pour optimiser le résultat. L'automatisation du flot de conception répond à une volonté d'accélérer la conception et de réduire les coûts, et à une nécessité d'adopter une méthodologie systématique pour gérer la complexité et améliorer la qualité du système. Au-delà d'une certaine complexité, l'expérience et l'intelligence du concepteur ne suffisent plus à maîtriser efficacement la conception de tels systèmes, qui impose de tenir compte de nombreux paramètres et de faire des compromis. Il serait donc plus efficace de tester systématiquement chaque possibilité.

Même si l'on était capable de générer en un temps raisonnable toutes les possibilités de systèmes respectant la spécification de départ, il resterait encore à évaluer la meilleure solution. Or simuler un système au niveau RTL ou bien le prototyper prend un temps non négligeable, et ne peut pas être fait un grand nombre de fois. Il est donc nécessaire de faire des choix à chaque étape afin de guider le processus d'exploration. Pour cela, il est nécessaire d'estimer l'influence sur les performances de chaque décision. L'architecture peut notamment être évaluée en utilisant des modèles de simulation de haut niveau [Ros04]. Les travaux de cette thèse vise le raffinement de l'architecture en un modèle RTL et font l'hypothèse que l'allocation des ressources et la synthèse des communications sont réalisées par le concepteur.

1.4.2 Objectifs

a) Optimiser le système

Des modèles de haut niveau existent pour estimer les performances de l'architecture d'un système avant le développement d'un modèle RTL. Ces modèles ayant un haut niveau d'abstraction, il reste un certain nombre de choix de conception à prendre au moment de la réalisation du modèle RTL. Certains choix ne sont en effet pas évaluables à haut niveau. Par exemple, il peut être difficile de choisir entre deux bus celui qui est optimal en termes de performances, tant qu'un modèle précis au cycle près n'a pas été évalué.

De même, la pertinence de certains choix dépend du raffinement de l'architecture. Les performances du protocole dépendent notamment de l'interface de communication. L'évaluation des performances de la communication est moins précise avant la réalisation du modèle RTL. Si l'on veut envisager différents protocoles de communication, il est donc nécessaire de générer des modèles RTL du système rapidement. Il n'est donc pas optimal de fixer définitivement l'architecture avant la réalisation du modèle RTL.

Développer des outils permettant de raffiner automatiquement une architecture en un modèle RTL est nécessaire pour explorer les différents choix de conception qui s'offrent au concepteur. Etant donné que la conception des interfaces de communication prend du temps et que les interfaces de communication sont difficilement réutilisables, automatiser leur conception est nécessaire pour réduire le délai entre un modèle abstrait du système et un modèle RTL, et optimiser le système. Le concepteur système peut alors changer des composants, modifier le réseau ou changer les paramètres du système, puis rapidement générer un nouveau modèle du système au niveau RTL.

b) Optimiser l'interface

Avec la complexité croissante des applications, les systèmes monopuces requièrent des réseaux de communication plus performants. Dans le même temps, interconnecter tous les composants d'un système monopuce devient problématique (fiabilité des interconnexions, temps de propagation des signaux, place occupée par les interconnexions). Ces deux tendances rendent les réseaux de

communication de plus en plus compliqués. Dans ce contexte, les interfaces de communication ont un rôle important. Elles doivent évoluer pour supporter des protocoles de communications plus compliqués.

Or les ressources de communication sont allouées en fonction du type d'application visée par le système (bande passante pour applications vidéo, fiabilité des communications pour un système de freinage …). Par conséquent, ces interfaces doivent être adaptées aux différents besoins de l'application (en bande passante, en latence, au modèle de communication : mémoire partagée/passage de messages). Selon le protocole de communication, certaines interfaces sont mieux adaptées. Pour une communication par mémoire partagée, il est par exemple probablement plus performant de concevoir l'interface en concordance avec le contrôleur mémoire pour qu'elle puisse faire des accès directs à la mémoire sans utiliser le processeur [Cul99b]. Des interfaces réutilisables ne sont pas toujours optimisées pour les besoins du système. En conséquence, explorer différentes interfaces est intéressant pour optimiser les communications. L'automatisation de la conception des interfaces de communication répond également à cet objectif.

De plus, le développement du logiciel embarqué prend une place croissante dans la conception des systèmes monopuces. Cette tendance est accentuée par l'utilisation de protocoles de communication plus évolués. Développer du matériel efficace et fiable n'est pas suffisant si le logiciel n'est pas capable de l'utiliser efficacement. Evaluer les performances d'une interface requiert de tenir compte de son interaction avec le logiciel. Au niveau du logiciel, on sépare généralement le code de l'application du code responsable de la gestion des ressources matérielles et du contrôle de la communication. Le contrôle de la communication est alors réalisé en logiciel par des fonctions particulières appelées pilotes de communication.

Le contrôle de la communication étant supporté à la fois par les pilotes logiciels et l'interface de communication, certaines fonctionnalités peuvent être réalisées aussi bien dans le pilote que dans l'interface. Les performances des communications dépendent bien évidemment de la réalisation choisie. Optimiser les communications passe donc par l'accélération matérielle des communications. Le contrôle de la communication uniquement en logiciel pourrait être moins efficace et rendre le développement logiciel plus compliqué. Un objectif de cette thèse est d'adapter les performances des communications au système en permettant d'évaluer différents choix de réalisation logiciel/matériel du protocole de communication.

c) Aide au développement des interfaces

L'intégration logiciel/matériel est un problème important dans les systèmes monopuces. Modéliser le système à haut niveau aide à gérer l'hétérogénéité du système mais l'intégration du logiciel et du matériel reste une tâche difficile. Le développement des pilotes et des interfaces de communication est généralement fait par deux équipes différentes, découvrant les erreurs quand le pilote est exécuté sur le matériel. Une mauvaise interprétation de la frontière entre le matériel et le logiciel, ou une

documentation mal définie est une source d'erreurs. A cause du temps de développement croissant du logiciel, développer le matériel et le logiciel en parallèle est nécessaire et les problèmes d'intégration logiciel/matériel deviennent de plus en plus importants. Un des objectifs des travaux de cette thèse est de faciliter l'intégration logiciel/matériel. La génération automatique des interfaces rend l'intégration logiciel/matériel plus facile. Les travaux de cette thèse ont été menés dans le cadre d'un travail d'équipe visant à la réalisation d'un flot de conception système complet.

De plus, l'effort de conception des interfaces de communication croit avec la complexité croissante de la communication dans les systèmes monopuces. Automatiser leur conception est donc d'autant plus intéressant dans la mesure où cela permet un gain de temps et d'argent.

1.5 Contributions

1.5.1 Modélisation du protocole de communication

La première contribution de cette thèse est l'adaptation d'un modèle de protocoles développé pour les réseaux au domaine des systèmes monopuces. En effet, si des modèles de protocoles de haut niveau sont couramment utilisés dans les réseaux, ils n'ont fait leur apparition que très récemment dans le domaine des systèmes monopuces. Ces modèles de communication sont intéressants pour aider le concepteur à maîtriser la complexité des systèmes grâce à leur haut niveau d'abstraction.

Ces modèles sont généralement utilisés pour modéliser les communications et structurer l'architecture du réseau. Néanmoins, la structure et le haut niveau d'abstraction de ces modèles les rendent mal adaptés à une réalisation matérielle. Les interfaces de communication sont généralement conçues et optimisées à la main (à partir d'une spécification papier). Dans les réseaux, on utilise donc ces modèles pour décrire une architecture matérielle déjà conçue. Dans notre cas, nous souhaitons spécifier les interfaces à l'aide d'un tel modèle et générer les interfaces à partir de ce modèle.

Un modèle de protocoles a donc été adapté dans le cadre de cette thèse pour pouvoir servir à la conception des interfaces de communication, et permettre son raffinement automatique en une réalisation physique. Notre méthodologie de conception des interfaces de communication part d'une description des services de communication requis par l'application et des services offerts par le réseau. Le concepteur doit alors modéliser le protocole ou le choisir parmi une liste de protocoles déjà modélisés. L'analyse des différents protocoles et la sélection du meilleur protocole ne fait pas partie des travaux abordés par cette thèse, mais l'utilisation d'un modèle de haut niveau facilite la modélisation du protocole et l'expérimentation de plusieurs choix de protocoles.

1.5.2 Le découpage logiciel/matériel de la communication

Le respect du protocole de communication concerne à la fois les pilotes de communication et l'interface de communication. Le contrôle de la communication étant de la responsabilité commune de l'interface et des pilotes, il existe différentes solutions de réalisations en terme de conception en

matériel ou en logiciel. Cette thèse aborde partiellement le problème de l'exploration de différents choix de réalisation logiciel/matériel afin d'optimiser les communications. Dans cette perspective, les travaux de cette thèse n'ont pas visé à définir une méthodologie pour choisir les fonctionnalités à réaliser en matériel ou logiciel, mais à proposer un modèle pouvant être utilisé par le concepteur pour faire facilement ce choix.

Le découpage logiciel/matériel est réalisé sur le modèle du protocole. Le modèle du protocole sert pour spécifier à la fois les pilotes et l'interface de communication. On peut donc l'utiliser pour décrire l'ensemble du système de communication, ce qui facilite l'intégration logiciel/matériel en évitant des problèmes d'incohérence entre les pilotes de communication et l'interface ainsi que des problèmes de non respect du protocole.

Une des contributions de cette thèse est d'avoir défini comment on pouvait obtenir une spécification de l'interface à partir du modèle de protocoles découpé. La génération des interfaces de communication s'inscrit alors dans un flot global développé au sein du laboratoire TIMA-SLS, qui permet également de générer les pilotes de communication à partir de leur spécification obtenue après le découpage logiciel/matériel.

1.5.3 Génération automatique des interfaces de communication

La troisième contribution de cette thèse est une méthodologie de génération automatique des interfaces de communication. La méthode prend en entrée une spécification obtenue à partir du découpage logiciel/matériel et permet d'obtenir un modèle RTL synthétisable. L'approche s'appuie sur un assemblage systématique de composants de base issus d'une bibliothèque. On fournit un environnement d'assemblage, et une aide à l'assemblage des composants de bibliothèque. Un outil a été développé à partir de la méthodologie définie afin de réduire le temps de conception des interfaces.

On accélère le processus de conception des interfaces car l'écriture de la spécification est plus rapide et plus aisée que l'écriture d'un modèle RTL. On facilite de plus l'intégration des composants du système car la méthodologie est intégrée dans un flot global de conception des systèmes monopuces.

1.6 Plan du mémoire

Ce manuscrit est organisé en six chapitres. Le chapitre suivant commence par une étude des communications dans les systèmes monopuces, et replace la génération des interfaces dans le problème de la réalisation des communications. Cela permet d'amener l'analyse des méthodes existantes pour la génération automatique de ces interfaces, afin de cerner en quoi elles remplissent ou ne remplissent pas nos objectifs et nos contraintes. Le chapitre 3 présente un modèle du protocole de communications qui est découpé en une partie logicielle et une partie matérielle afin d'obtenir la spécification de l'interface. Le chapitre 4 expose le principe de la génération automatique d'un modèle synthétisable de l'interface, et son application à la réalisation d'un outil. Le chapitre 5 décrit deux

expérimentations de la méthodologie de génération des interfaces, et s'intéresse à la validité et à l'efficacité de la méthode. Le dernier chapitre conclut en rappelant le travail réalisé et ses limites, et dresse un bref tableau des perspectives offertes par ce travail.

Chapitre 2

Conception d'interfaces de communication pour des systèmes monopuces

Sommaire

2.1 Introduction

Après avoir expliqué pourquoi il est intéressant de réduire le temps de conception des interfaces de communication, ce chapitre étudie maintenant les méthodes de conception existantes. Pour comprendre le problème de la conception des interfaces, il est important d'aborder le problème de la communication dans les SoC dans sa globalité. La communication entre les unités de traitement se fait à travers un réseau de communication. Les interfaces de communication connectent pour cela les unités de traitement au réseau de communication.

Ce chapitre présente en premier un exemple de SoC et les problèmes liés à la communication dans ce système afin d'exposer les problèmes que nous souhaitons analyser. Un bref état de l'art sur les composants utilisés dans un système monopuce permet de définir le contexte de l'étude (que veut-on faire communiquer et sur quel type de réseau), ceci afin d'introduire le problème de la communication dans les systèmes monopuces et le rôle des interfaces de communication qui est analysé. Un état de l'art des méthodes existantes de conception des interfaces de communication justifie les objectifs et les hypothèses fixées par cette thèse.

2.2 Description d'un SoC industriel : le système Nexperia

Les systèmes monopuces deviennent de plus en plus complexes pour supporter de nouvelles applications telles que les applications multimédia. Le système monopuce *Nexperia pnx 8525* (surnommé "Viper") [Phi01] est un exemple de systèmes destiné à des applications vidéos et à la télévision numérique ("set-top box" en anglais). Cet exemple de système monopuce a pour objectif de montrer la complexité des communications dans les systèmes actuels et leurs évolutions. [Dut01] a discuté des choix de conception adoptés et des difficultés rencontrées, et donne un aperçu de l'expérience acquise avec le développement de ce circuit. [Goo04] a réalisé une analyse des communications dans ce système, dont les résultats sont brièvement exposés ici.

Un tel système doit non seulement fournir de fortes puissances de calcul pour supporter les fonctions d'encodage/décodage vidéo et de traitement de l'image, mais doit aussi posséder un réseau de communication sophistiqué pour supporter les différents flux de données aussi bien audio que vidéo gérés par le système.

Le système Nexperia est construit autour d'un processeur général MIPS-PR3940, et d'un processeur dédié au traitement du signal TriMedia TM3218 (figure 2.1). Il ne possède pas de mémoires internes mais accède à une unique mémoire externe par l'intermédiaire d'un contrôleur mémoire (cette mémoire n'est pas dessinée sur la figure 2.1). Le système est également composé d'un certain nombre (~50) de périphériques à usage général et de coprocesseurs matériels utilisés pour le traitement de l'image, ou bien dédiés aux entrées/sorties vidéo.

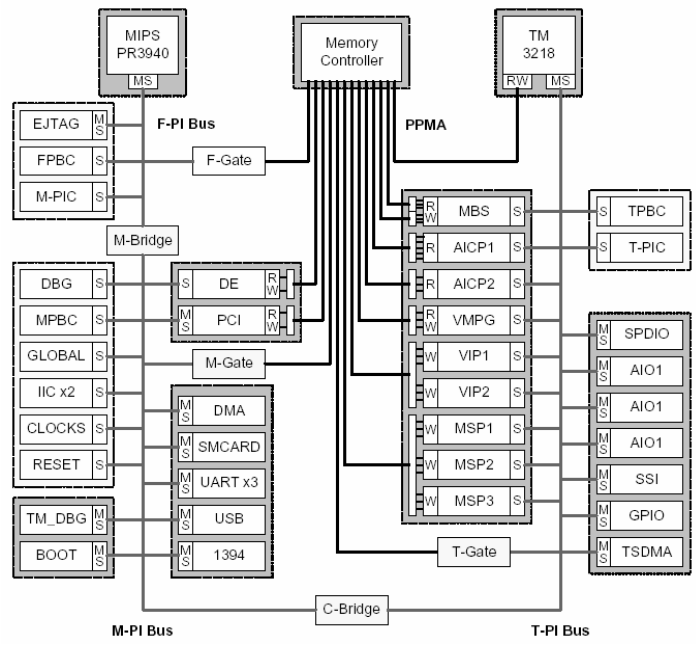


Figure 2.1 - L'architecture du système Nexperia (extrait de [Goo04])

Comme un réseau à un seul bus partagé n'aurait pas supporté tout le trafic nécessaire (avec les contraintes de performances demandées), une solution basée sur plusieurs bus et des liaisons point à point a été retenue. La difficulté de la conception d'un tel réseau est de l'adapter à différents types de trafic avec des caractéristiques différentes en termes de débit, latence et "jitter" (variation dans le débit des données). Cela est d'autant plus ardu qu'il n'est pas évident d'évaluer les performances à fournir car le trafic dépend fortement de l'application, et les différents modes de fonctionnement du circuit (statique et dynamique) requièrent des communications fortement configurables.

Le transfert des données entre les blocs est réalisé par mémoire partagée, comme l'illustre la figure 2.2. Ce système nécessite d'importantes ressources mémoires à cause de la taille des images manipulées. En raison du coût des mémoires embarquées et de la taille des données échangées, une mémoire externe a été utilisée.

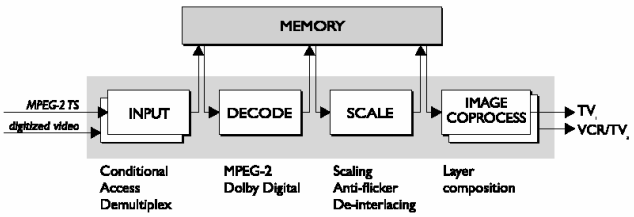


Figure 2.2 - Communication par mémoire partagée (extrait de [Phi01])

Les interconnexions ont été conçues de manière à séparer le trafic ayant un bas débit du trafic ayant un haut débit pour lui assurer une faible latence. Deux PI-bus [OMI], reliant les IP aux deux processeurs, supportent le trafic de contrôle, qui nécessite une bande passante assez faible. Les bus partagés connectant les IP servent uniquement au contrôle ou aux transferts des données de petite taille. Des connexions point à point 64 bits ont du être utilisées pour supporter les communications

avec le contrôleur mémoire, en offrant les bandes passantes requises. Le processeur MIPS est connecté au contrôleur mémoire par un pont pénalisant la latence des accès à la mémoire.

Ce réseau de communication a été conçu sur mesure pour ce système afin d'optimiser ses performances et son coût. La structure et le type des interconnexions ont été choisis en fonction des besoins spécifiques du système. Le choix des politiques d'arbitrage est aussi très important, mais a été difficile dans ce système en raison de l'interaction des différents arbitres des bus et du contrôleur mémoire ; les différents bus et le contrôleur mémoire étant tous connectés entre eux par des ponts (adaptateurs de bus, ou "bridge" en anglais).

La figure 2.3 montre l'architecture du système Viper 2, qui est une évolution du système précédent. Au niveau des unités de traitement de données, la principale évolution par rapport à Viper est l'utilisation de deux processeurs TriMedia et non plus d'un seul (les versions des processeurs ne sont également plus les mêmes). Quelques IP ont été ajoutés mais on retrouve à peu près les mêmes IP que dans le système précédent avec des puissances de calcul plus importantes. Le réseau de communication a été revu en conséquence pour offrir de plus grandes performances afin de supporter un trafic plus important.

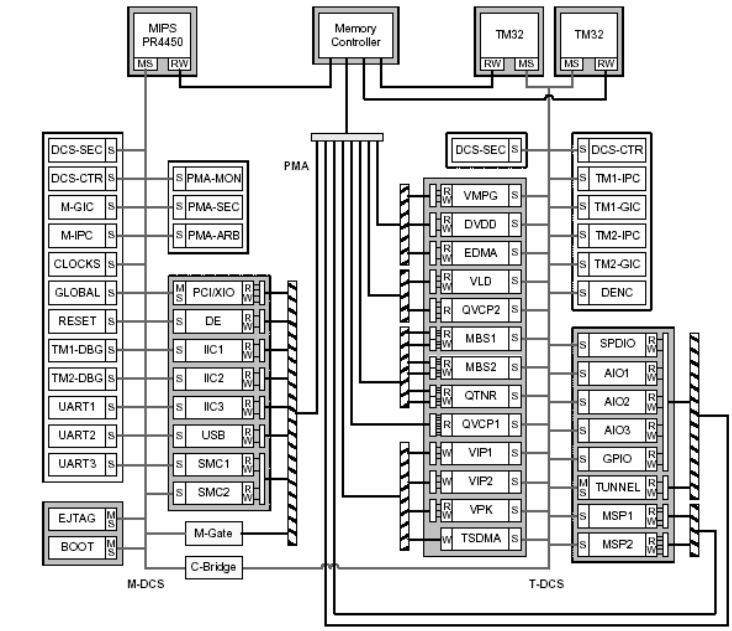


Figure 2.3 - L'architecture du système Viper2 (extrait de [Goo04])

L'utilisation de PI-Bus dans Viper avait été contrainte par la réutilisation de blocs IP disponibles. Cela illustre les limites d'une telle méthodologie que nous avions évoquées dans le chapitre 1.3. Le pont entre le PI-Bus du MIPS et le contrôleur mémoire pénalisait par exemple les performances. Les problèmes rencontrés dans Viper, qui ont entraîné le besoin de faire évoluer le réseau de communication, mettent en évidence le principe énoncé au chapitre 1 du découplage du calcul et de la communication.

L'expérience de Viper a motivé la création de blocs IP indépendants de l'interconnexion dans Viper 2 afin de faciliter la réutilisation des IP et des interconnexions, et leurs évolutions. C'est ce qui a été fait avec l'utilisation du standard de protocole DTL.

Les PI-Bus ont été remplacés par des bus DCS (bus dédié au contrôle de composant : faible débit, faible latence). L'utilisation de ce bus (transaction d'un seul mot), combiné avec une très faible charge (0.5%), assure une faible latence. Dans le système Viper2, les accès aux IP se font avec un mécanisme de lecture et d'écriture postées [Goo04, Lys02] (la fréquence d'horloge des IP étant plus faible que celles des processeurs et des bus DCS, la latence des accès aux blocs IP peut être optimisée en accédant à une copie des registres se trouvant dans l'interface). L'utilisation combinée du bus DCS et du mécanisme de lecture et d'écriture postées a permis de réduire les temps de latence par rapport à Viper malgré un nombre d'IP plus grand. Cela illustre parfaitement l'intérêt des adaptateurs pour réutiliser les composants tout en optimisant les communications. Les modifications nécessaires à cette optimisation sont alors restreintes aux adaptateurs.

Pour supprimer les problèmes d'interactions entre les arbitres, le partitionnement des interconnexions s'est fait en fonction du type de trafic, évitant d'utiliser des ponts entre les différentes interconnexions. Les connexions point à point de Viper ont été mises en cascade (facilite le placement/routage et évite les problèmes d'interaction du Viper en séparant les différents types de trafic à l'entrée du contrôleur mémoire). Des adaptateurs entre les connexions point à point et le contrôleur mémoire avaient déjà été utilisés dans Viper, facilitant ainsi la réutilisation des IP dans Viper2, tout en ne nécessitant que des modifications mineures de l'adaptateur. Tout comme dans Viper, le trafic avec des caractéristiques similaires (en débit, latence et "jitter") est regroupé dans les adaptateurs avant d'entrer dans les connexions point à point, avec un premier niveau d'arbitrage. Dans le cas des adaptateurs entre les IP et les connexions point à point, des interfaces avec de la mémoire étaient également nécessaires pour formater les données en trame de 128 octets, gérer la communication entre plusieurs domaines d'horloges, et pour servir de mémoires tampon (afin de s'affranchir du "jitter" introduit par la latence variable des accès mémoires).

La présentation de cet exemple a permis de mettre en valeur l'intérêt des interfaces de communications dans les systèmes monopuces. Elle a également montré que ces interfaces devaient offrir différentes fonctionnalités, et devaient être adaptées au système. Leur relative complexité est donc un des problèmes de la conception des systèmes monopuces.

2.3 Evolution et tendance des systèmes monopuces

Avant de rentrer plus en profondeur dans l'analyse du problème des communications, il est nécessaire de dresser un bref aperçu des composants que l'on connecte dans un système monopuce et comment on les connecte.

2.3.1 Les unités de traitement de données

a) CPU

Comme nous l'avons vu dans le premier chapitre, les systèmes monopuces intègrent des processeurs à usages généraux [MIPS, ARM]. Les premiers processeurs utilisés dans les systèmes monopuces étaient relativement basiques, mais ils ont évolué pour fournir les performances requises par des applications gourmandes en puissance de calcul. Pour accélérer l'exécution de leurs programmes, les processeurs utilisent maintenant des mémoires caches et des architectures "pipeline", une gestion de la mémoire améliorée, des mécanismes améliorés pour le partage de la mémoire et la synchronisation entre plusieurs processeurs [Bra02]. Les jeux d'instructions de ces processeurs comprennent des instructions dédiées au traitement du signal ou à l'accélération matérielle de programmes JAVA.

On peut également retrouver dans les systèmes monopuces des processeurs à usages spécifiques tels que des processeurs dédiés au traitement du signal (DSP) [Eij99].

Certains processeurs [TEN] peuvent être configurés pour une application spécifique, et leurs jeux d'instructions classiques peuvent être étendus par l'utilisateur avec des instructions spécifiques par ajout de logiques dans le cœur du processeur.

b) Mémoires

Des mémoires internes offrent de meilleures performances que des mémoires externes en termes de latence et bande passante. [ITRS] (cité par [Gha03]) prévoit que la surface de silicium occupée par les mémoires internes va devenir prépondérante dans les années à venir. Par conséquent, la taille de la mémoire va donc représenter un critère important pour le coût du système, ainsi qu'une contrainte technologique. Avec l'évolution des circuits, on a besoin à la fois de mémoires de plus en plus grandes, mais aussi d'une bande passante de plus en plus élevée. On peut distinguer plusieurs types de mémoires (SRAM, SDRAM, ROM …) avec des propriétés, des performances et des coûts différents. Les choix des mémoires et leur organisation sont des problèmes difficiles à résoudre. Utiliser une seule grande mémoire est avantageux en terme de surface par rapport à l'utilisation de plusieurs mémoires plus petites (coût du contrôleur mémoire plus faible). En revanche, l'utilisation de plusieurs mémoires plus petites est plus intéressante en termes de bande passante et de latence [Cul99b]. Au moment du choix des mémoires, il faut également tenir compte des mémoires caches et dimensionner celles-ci.

c) IP matériels

On emploie le terme IP pour parler d'un composant matériel réutilisable. L'utilisation d'IP pour la conception des systèmes monopuces s'impose aujourd'hui comme une voie privilégiée, et l'offre d'IP disponibles ne cesse de se développer [D&R]. On trouve de nombreux types d'IP à usage général (ex. : contrôleur d'interruptions), pour les entrées/sorties (ex. : contrôleur liaison série), le calcul (ex. :

encodeur vidéo), la communication (ex. : bus partagé), et les mémoires qui peuvent être également considérées comme des IP. Bien que la réutilisation de blocs matériels doive réduire les temps de conception, leur utilisation est contrainte par les difficultés de leur intégration dans un système. L'utilisation d'un IP conçu par une autre équipe de travail implique d'avoir confiance en cette équipe, et requiert une bonne connaissance de l'IP. Pour faciliter l'utilisation des IP et leur intégration dans un système, des méthodes et des standards sont en cours de développement. [Spi04] a défini un format pour la description et l'empaquetage d'IP, en vue du développement d'outils d'intégration d'IP ("Structure for Packaging, Integrating and Re-using IP with Tool-flows"). [VSIA] travaille sur des standards de qualité, ainsi que des méthodes de protection et d'échange des IP.

2.3.2 Des interconnexions jusqu'aux réseaux sur puces

a) Topologies

Comme nous l'avons vu dans l'exemple du chapitre 2.1, les performances d'un système dépendent fortement de son réseau de communication. Choisir la structure du réseau de communication est une des difficultés majeures de la conception d'un système. Plusieurs topologies peuvent être employées pour le réseau : connexions point à point, bus partagés interconnectés par des adaptateurs, réseau en anneau ("token ring"), "crossbar", réseau sur puce. Le choix se fait en fonction des performances requises par l'application. Différents types de réseaux offrent différents types de performances, et les performances d'un réseau dépendent du type de transferts réalisés par l'application [Lah00]. Les critères de décision pour le choix d'un réseau sont la bande passante offerte, la latence des communications, le coût physique du réseau, sa consommation, sa flexibilité (possibilités de configuration). La structure du réseau doit être choisie avec d'autant plus d'attention que des réseaux ayant une bande passante élevée ont aussi une latence élevée en général. Des compromis entre les différents critères de choix sont généralement nécessaires et doivent être réalisés en fonction de l'application. Même si l'on cherche à avoir des réseaux de communication spécifiques, il est également nécessaire de les réutiliser pour diminuer les temps de conception. Les possibilités de configuration du réseau permettent de l'adapter au système, et sont donc un paramètre important.

b) Les bus partagés

La solution la plus intéressante en terme de partage de ressources est le bus partagé. Pour offrir de grandes performances, les protocoles de bus offrent des mécanismes de transfert de paquets de données (transfert en rafale, ou "burst operation" en anglais), des mécanismes de "split" (interruption de la transmission par l'esclave), de verrouillage du bus, de réponse.

Dans le cas de bus supportant plusieurs maîtres, il est nécessaire d'arbitrer l'accès de chaque maître au bus. Le choix de la politique d'arbitrage est généralement un choix de réalisation, et n'est pas fixé dans la spécification du bus. Celle-ci doit être précautionneusement étudiée comme nous l'avons vu dans le cas de l'exemple Viper, en fonction de l'ordonnancement des communications.

En fonction de l'utilisation visée par le bus (exemple : bus local au processeur, ou bus pour des périphériques), les contraintes et les performances requises ne sont pas les mêmes. [Fly97, ARM99] et [IBM99] proposent à cette fin des familles de bus avec différents compromis coût/performance/simplicité.

Les inconvénients des bus sont une bande passante limitée, et un nombre limité de composants pouvant être connectés. Les bus ne sont donc pas extensibles. Il faut également tenir compte du fait que plus la charge du bus est importante, plus la latence moyenne des transactions est élevée.

Afin de s'affranchir de ces limites, les systèmes utilisent généralement plusieurs bus connectés entre eux par des ponts. L'inconvénient d'une telle solution est que les ponts apportent une pénalité en performance, car ils rajoutent un temps de latence supplémentaire.

c) Les réseaux- sur- puces

Les réseaux sur puce, ou NoC ("Network-On-Chip" en anglais) [Wie02, Dal01], adoptent une structure calquée sur celle des réseaux d'ordinateurs. Les NoC sont construits à partir de routeurs connectés entre eux par des interconnexions point à point. Ces routeurs s'occupent du routage de paquets de données sur les fils. Les paquets ont généralement des en-têtes, contrôlant le routage des données.

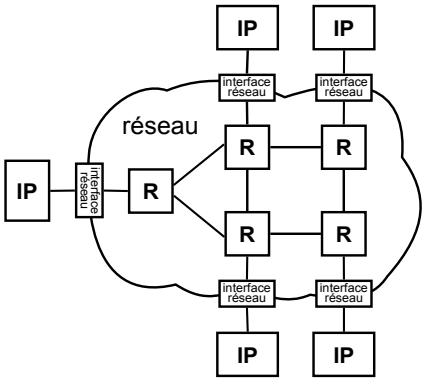


Figure 2.4 - Schéma simplifié d'un réseau sur puce [Rad03]

De nombreux NoC ont été développés aussi bien dans le domaine de la recherche [Gue00, Jal04, Jan03] que dans le domaine industriel [Die03, Kar01, Arteris, Win01].

2.4 Problématique de la communication et rôle des interfaces de communication

2.4.1 Paradigme des réseaux d'ordinateurs

Comme nous venons de le voir, les réseaux de communication ont fortement évolué afin de répondre aux contraintes physiques de leur conception et aux performances requises par l'application. Les temps de communications ne sont plus négligeables, et cela représente un changement important pour la conception des systèmes. Les communications risquent d'être un facteur limitatif pour les performances du système. Le problème est similaire à celui rencontré dans les machines parallèles. Le

parallélisme est étroitement lié à la communication, car le temps gagné par l'exécution parallèle d'une opération ne doit pas être perdu à cause de la communication entre les différentes machines réalisant cette opération. [Alm89] (cité par [Cul99a]) décrit une machine parallèle comme étant "un ensemble d'éléments de calcul qui communiquent et coopèrent pour résoudre de gros problèmes plus rapidement". Cette définition est très proche de celle qui a été donnée d'un système monopuce dans le chapitre 1.3.2 : "un ensemble de nœuds de calcul logiciel (processeur) ou matériel (IP) communiquant à travers un réseau de communication".

Une coopération efficace de l'ensemble des composants d'un SoC ou d'une machine parallèle est étroitement liée à la communication, qui doit donc être étudiée avec autant de soin que l'algorithme, son découpage en tâches parallèles, ou que le choix des composants (logiciel ou matériel). Les machines parallèles utilisent donc la notion d'architecture de communication, définissant les communications de base, les opérations de synchronisation, ainsi que l'organisation de la structure qui réalise ces opérations [Cul99a].

Pour fournir les performances requises pour un coût le plus faible possible, cette architecture de communication doit être ciblée pour un certain type d'applications. A la différence des machines parallèles qui sont généralement homogènes, les SoC sont hétérogènes. L'hétérogénéité du système et la nécessité d'avoir des systèmes flexibles et évolutifs rendent difficiles les approches globales et régulières. Nous avons vu dans le chapitre 2.2 l'exemple de l'architecture Viper où les différents types de trafic dans l'architecture Viper ont été séparés pour assurer les performances requises.

[Ben02] présente les réseaux sur puce comme un nouveau paradigme de conception des systèmes monopuces, où il n'existe pas de synchronisation globale des communications, et où chaque composant communique et initie des transferts de façon autonome. Comme nous l'avons vu dans le cas du système Nexperia au chapitre 2.2, la synchronisation des communications est une tâche difficile. Il est plus facile de la gérer en la décomposant grâce à un contrôle de la communication distribué. Un réseau d'ordinateurs étant un ensemble interconnecté d'ordinateurs autonomes [Tan96]. Cette autonomie amène à faire le rapprochement entre un système monopuce et un réseau d'ordinateurs, on ne parle plus de systèmes monopuces mais de réseaux monopuces, ou NoC.

2.4.2 Hiérarchie de protocoles et le modèle de référence OSI

Dans la mesure où l'on adopte le paradigme des réseaux pour la conception des systèmes monopuces, il convient de s'intéresser à la façon dont les communications sont modélisées dans les réseaux. Afin de gérer la complexité de la conception des réseaux, ceux-ci sont la plupart du temps structurés en couches. L'objectif est de décomposer le problème en construisant chaque couche sur la précédente. Chaque couche offre une abstraction des couches inférieures et du support physique à la couche supérieure. La figure 2.5 montre un exemple de modèle en couches.

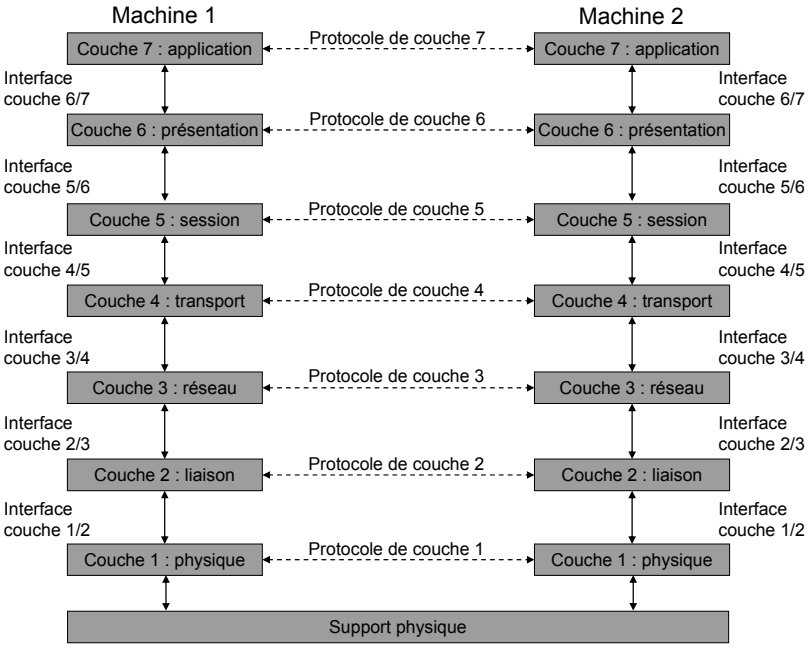


Figure 2.5 - Couches, protocoles et interfaces

Pour chaque couche N d'une machine, les données sont transférées d'une machine à l'autre par la couche N-1. Tout se passe donc comme si on avait une communication directe entre chaque couche d'une machine. On parle de communication virtuelle pour qualifier la communication entre chaque couche N d'une machine. En réalité, la communication se fait en transférant les données d'une couche à la couche immédiatement inférieure jusqu'à la couche la plus basse qui transmet les données sur un support physique.

L'organisme international de normalisation ISO a défini un modèle de référence ou modèle OSI [Tan96]. Le modèle OSI distingue sept couches, et définit leurs rôles respectifs (figure 2.5). Les couches physique, liaison de données et réseau s'occupent du transfert de données sur le réseau de communications (support physique + routeurs), et permettent le transfert de données d'une machine à une autre machine. Les couches transport, session, présentation et application sont elles liées à l'application. Les rôles des différentes couches sont les suivants :

- Physique : transmission “brut” des bits sur un canal de communication
- Liaison : fiabilisation de la transmission, gestion de l'accès à un canal partagé, régulation du trafic (pour éviter de saturer le récepteur)
- Réseau : acheminement des paquets de la source au destinataire (routage), gestion des congestions
- Transport : fiabiliser la transmission, établissement et fermeture des connexions, contrôle de flux, fragmentation des messages en paquets, masquer le sous réseau (on parle de couche de bout en bout, c'est-à-dire de l'émetteur au destinataire)
- Session : transfert de fichier, accès partagé à des ressources informatiques, synchronisation
- Présentation : gestion de la syntaxe et de la sémantique des données échangées
- Application : gestion des communication entre des applications communicantes (web, messagerie, terminal virtuel …)

Les rôles de chaque couche du modèle ne seront pas développés plus longuement ici dans la mesure où l'application directe du modèle OSI aux systèmes monopuces semble mal adaptée pour des raisons qui sont explicitées plus loin. Un point central du modèle OSI est la définition et la distinction de trois concepts pour la structuration d'un protocole en couches : les services, l'interface et le protocole.

Pour que la communication puisse s'établir entre les couches N de deux machines, les couches N de chaque machine doivent employer le même protocole de communication. Le protocole de communication d'une couche N est un ensemble de règles à respecter concernant les opérations de l'échange, le format et la signification des données échangées entre les couches N de deux éléments du réseau. L'ensemble des protocoles utilisés par toutes les couches constitue la pile de protocoles du système. Le modèle OSI définit les rôles de chaque couche mais pas leurs protocoles.

Une couche est un fournisseur de services pour la couche immédiatement supérieure. Un service définit une fonctionnalité offerte par la couche N à la couche N+1, mais ne fait aucune hypothèse sur la manière de réaliser cette fonctionnalité, ni sur le protocole de la couche N. La frontière entre deux couches adjacentes est appelée l'interface des deux couches. On accède aux services par l'interface. Chaque couche étant réalisée sur la précédente, il est nécessaire que l'interface de deux couches adjacentes cache le contenu de chaque couche. Ceci afin d'abstraire la couche inférieure, pour cacher sa complexité à la couche supérieure qui l'utilise. L'interface entre deux couches adjacentes définit pour cela les services que la couche inférieure fournit à la couche supérieure, mais aussi comment on y accède (paramètres à utiliser, type de résultat à attendre). Chaque machine peut utiliser indifféremment n'importe quelle interface, tant que chaque couche respecte le bon protocole. L'interface est composée de Points d'Accès aux Servies, ou SAP (Service Access Point) par lesquels la couche N+1 peut accéder aux services de la couche N. Un SAP peut regrouper plusieurs services. Le service est un concept abstrait. On le spécifie donc par un ensemble de primitives (opérations), permettant d'accéder au service offert par une couche. OSI distingue quatre classes de primitives : les primitives de requête, d'indication, de réponse et de confirmation.

Le modèle OSI est un modèle abstrait des communications. Il décrit pour cela, ce que chaque couche doit réaliser. Il est indépendant de toute réalisation particulière, chaque couche pouvant être réalisée aussi bien en logiciel qu'en matériel, pour les couches les plus basses essentiellement. Pour plus de détails sur les piles de protocoles et le modèle OSI, le lecteur pourra se référer à [Tan96] qui en donne une description très complète.

2.4.3 Exemple d'application : le réseau Nostrum

Nostrum est un concept de plateforme de communication destiné à la conception de systèmes monopuces. La figure 2.6 décrit le principe de Nostrum. Dans ce projet, les différentes unités de traitement sont appelées des ressources. L'objectif de Nostrum est d'offrir une plateforme de communication fiable, pour supporter les différentes communications internes entre ces ressources.

Celle-ci peut-être adaptée aux performances requises par différents systèmes, et servir de base pour la construction du système en réutilisant le matériel des ressources qui deviennent des éléments du réseau. Cette infrastructure de communications est appelée “l'ossature du réseau Nostrum” (“the Nostrum Backbone” [Mil04]). Elle réalise les couches basses de la pile de protocoles, qui constituent la pile de protocoles Nostrum. En fonction des besoins spécifiques de l’application, une pile de protocoles spécifique est construite par-dessus cette pile. Cette pile est réalisée dans le RNI.

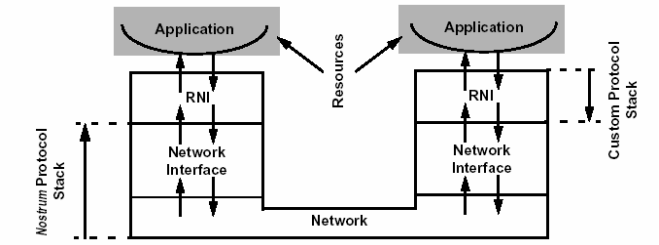


Figure 2.6 - L'ossature du réseau Nostrum [Mil02]

Le réseau et l'interface réseau (ou NI : “Network Interface”) constituent “l'ossature du réseau Nostrum”. Le terme réseau est ici employé pour décrire l'ensemble des routeurs et des interconnexions les reliant aux unités de traitement. Il doit être interprété au sens de réseau de communication, et exclut les unités de traitement. Il utilise une architecture maillée décrite à la figure 2.7.

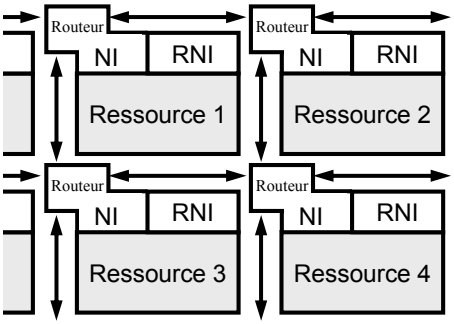


Figure 2.7 - L'architecture du réseau Nostrum [Mil04]

Le réseau et l'interface réseau utilisent la pile de protocoles Nostrum pour permettre la communication entre deux composants, considérés comme des ressources. En réalisant le protocole du réseau Nostrum entièrement dans l'interface, celle-ci permet d'offrir une abstraction du réseau aux ressources. L’interface réseau permet aux ressources de communiquer à travers le réseau, tout en leur offrant une interface uniforme. Deux types de services sont offerts par l'ossature Nostrum, des services avec une bande passante garantie, et des services sans garantie de performance. L’interface réseau s’occupe de la partie du protocole qui est propre au réseau.

Les unités de traitement sont soit connectées directement à l'interface réseau, soit connectées au RNI (“Resource Network Interface”) qui sert alors d'adaptateur entre l'unité de traitement et l'interface réseau. Le RNI peut ajouter une autre couche de protocoles par-dessus la pile de protocoles Nostrum et fournir ainsi à l’application d’autres services que ceux offerts par le réseau. Il permet donc d’offrir le modèle de programmation souhaité par l’application indépendamment du réseau utilisé. L'interface

réseau réalise la pile de protocoles Nostrum, le RNI réalise une pile de protocoles spécifique à l'application. Le RNI peut être réalisé en matériel ou en logiciel.

Le modèle OSI a été utilisé pour modéliser la pile de protocoles Nostrum. Les services et le rôle de chaque couche ont été définis dans [Mil02]. Cette pile comprend les couches physiques, liaison de données, réseau et transport du modèle OSI. La couche transport est facultative, au contraire des trois autres. Le modèle de Nostrum ne fixe ni les fonctionnalités réalisées dans le NI, ni celles dans le RNI. L'interface entre le NI et le RNI n'est pas figée. De même, le niveau d'abstraction des services offerts par le RNI et la profondeur de la pile de protocoles de l'application ne sont pas définis.

La modélisation du protocole du réseau Nostrum par une pile de protocoles est une aide à la conception du système mais ne fait aucune hypothèse sur la réalisation de l'interface de communication ; ceci afin de permettre d'optimiser la réalisation matérielle au niveau logique en regroupant certaines couches.

2.4.4 Un modèle en couches simplifié

Bien que les concepts du modèle OSI offrent des capacités de modélisation et d'analyse très intéressante, son application directe aux SoC semble mal adaptée [Ben02]. Le nombre important de couches du modèle OSI ne correspond pas à la complexité moindre des SoC par rapport aux réseaux d'ordinateurs, et à des contraintes différentes. [Ben02, Rad03] ont montré que les réseaux sur puces avaient des propriétés et des contraintes différentes des réseaux traditionnels qui se traduisaient par des choix de conception et de protocoles différents. La structuration de la pile de protocoles OSI ne correspond donc pas aux problèmes ciblés par les réseaux sur puce [Ben02]. Elle a été très fortement influencée par des besoins de standardisation, et des contraintes de compatibilité avec les infrastructures des réseaux existants. Ces réseaux étant bien souvent complètement découplés des applications finales. Les contraintes de compatibilité et de standardisation sont bien moins importantes dans un système monopuce, où le réseau de communication est développé par une unique équipe de conception. Les développeurs peuvent adapter le réseau de communication à une application, ou une classe d'applications, ciblée par le système.

En revanche, Nostrum introduit une idée intéressante avec la notion de pile de protocoles liée au réseau, et de pile de protocoles liée à l'application, même si aucune frontière n'est clairement fixée entre les deux. On peut retrouver ce principe dans [Pet03] qui donne un exemple de modèle simplifié d'un réseau, représenté à la figure 2.8.

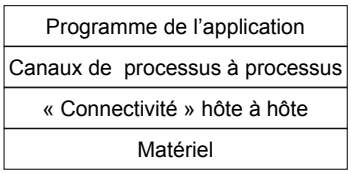


Figure 2.8 - Un modèle en couches simplifié d'un réseau [Pet03]

Nous définirons donc le problème de la communication dans les SoC en nous basant sur un modèle en couches. Ce modèle est un modèle d'étude et d'analyse, mais ne correspond pas à une réalisation comme dans le cas de Nostrum. L'application est définie comme un ensemble d'opérations sur des données. Le problème de la communication consiste à permettre l'échange de données entre deux parties de l'application. A cette fin, l'application utilise un réseau de communication. Pour utiliser les services du réseau, il est nécessaire d'utiliser une couche "hôte-à-hôte" qui va s'assurer de l'envoi des données sur le réseau et permettre le transfert de données entre deux nœuds de traitement des données. Si l'on se réfère au modèle OSI, celle-ci correspond aux couches physiques, liaison de données et réseau. Par-dessus cette couche peut également se trouver une couche permettant d'offrir des services de plus haut niveau d'abstraction à l'application.

Le problème de la communication dans les SoC présente malgré tout des différences par rapport au monde des réseaux. [Pet03] modélise la structuration du logiciel tandis que l'on souhaite modéliser un système hétérogène. Au lieu de parler de programme de l'application, on parle de tâche sur la figure 2.9 qui reprend la structure de [Pet03] avec une terminologie adaptée. L'ensemble de ces tâches constitue la spécification fonctionnelle du système. Chaque tâche peut être réalisée en matériel ou en logiciel. Le terme de machine hôte est lié au monde des réseaux. La couche "connectivité hôte-à-hôte" de la figure 2.8 a donc été renommée "couche d'accès au réseau" sur la figure 2.9. Le terme matériel est lui remplacé par "réseau de communication".

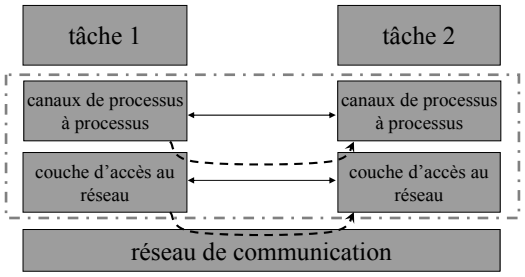


Figure 2.9 – Modèle en couches proposée

2.4.5 Rôle des interfaces de communication

Nous avons présenté précédemment un modèle en couches permettant de modéliser les communications, mais celui-ci n'est qu'un modèle de représentation. Il est donc nécessaire de comprendre comment passer d'un tel modèle à une réalisation d'un système monopuce.

Comme nous l'avons vu dans le chapitre 1, une interface de communication est un composant matériel connectant un sous-système processeur ou un IP à un réseau de communication. Les unités de

traitement vont donc utiliser leurs interfaces de communications pour communiquer entre elles à travers le réseau de communication. Il convient donc de situer cette interface par rapport au modèle des communications que nous venons de présenter.

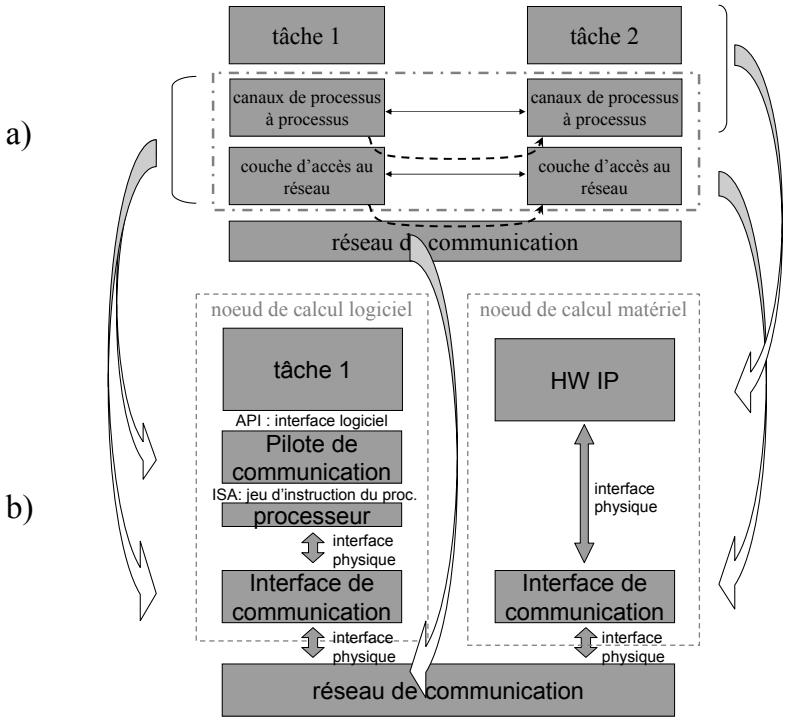


Figure 2.10 - Réalisation logiciel/matériel des communications

Dans le cas d'un nœud de calcul logiciel, la pile de protocoles est réalisée à la fois par un pilote de communication et par l'interface de communication. La figure 2.10 montre la manière dont le problème a été formalisé dans cette thèse. Dans le cas d'un nœud de calcul matériel, le composant matériel (IP) implémente généralement la couche processus-à-processus. Si ce composant matériel est relié par un bus, il faut alors utiliser une interface de communication entre l'IP et le bus. L'interface réalise alors la couche basse de la pile de protocoles.

Dans le cas d'un nœud de calcul logiciel, on a donc plusieurs choix de réalisation logiciel/matériel. Il faut choisir quelles fonctionnalités sont réalisées en logiciel et en matériel en fonction du compromis coût/performance désiré. Par exemple, [Bho03] a montré qu'une réalisation matérielle du découpage de message en paquets permettait d'obtenir un temps de latence environ 80 fois plus faible qu'avec une réalisation logicielle.

2.5 La conception des interfaces de communication

2.5.1 Les standards d'interfaces

Les interfaces de communication sont utilisées dans les SoC pour adapter les interfaces des composants de traitement avec celles des réseaux de communication. Le problème d'incompatibilité des interfaces est lié au fait que les composants de traitement et de communication sont conçus

indépendamment. Ce problème est donc résolu si des équipes de conception différentes s'accordent pour utiliser une même interface. Le problème est donc de définir un standard d'interface, définissant une interface commune pour tous les composants d'un SoC. L'intégration de tous les composants matériels d'un système monopuce devient alors très simple.

Une des premières initiatives a été lancée par le consortium *Virtual Socket Interface Alliance*, fondé par un groupe d'industriels désirant définir des standards pour la conception des systèmes monopuces [VSIA]. Ce groupe a établi un standard d'interface et un protocole standard, appelé *VCI* ("*Virtual Component Interface"*) [Len00, Lys00]. Un autre standard a vu le jour : le standard *OCP* de l'anglais "*Open Core Protocol*" [OCP03]. Crée par le consortium *OCP-IP* [OCP], il est en fait une évolution du standard *VCI* de *VSIA*. On citera également le standard *Whishbone* créé par *Silicore Corporation* [Ope02].

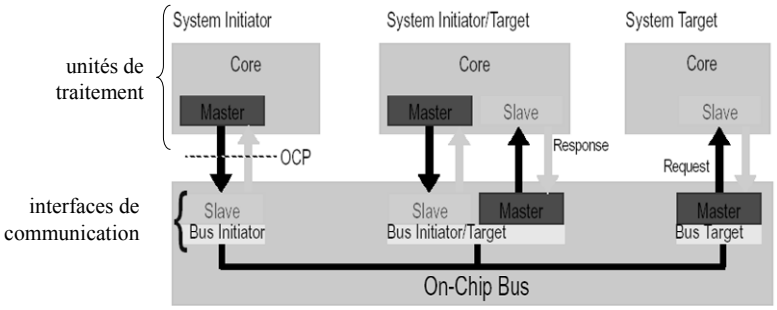


Figure 2.11 - La méthode de "socket" OCP [OCP03]

La figure 2.11 montre le principe d'un standard avec l'exemple de *OCP*. Les unités de traitement utilisent l'interface de *OCP*. Une interface de communication adapte le protocole *OCP* au protocole du bus. Les adaptateurs peuvent alors être réutilisés avec le bus. L'objectif de ces standards est la réutilisation des unités de traitement. Si l'on change de bus, il suffit alors de concevoir une nouvelle interface correspondant au nouveau bus. De même, il est facile de rajouter une unité de traitement dans le système car toutes les unités de traitement utilisent la même interface et le même protocole.

Le succès commercial de certains réseaux de communication, comme le bus *AMBA* [ARM99], a logiquement amené au développement d'IP pour ces réseaux. Le protocole de ces réseaux s'est donc imposé comme un standard de fait. Des interfaces réutilisables sont également disponibles pour adapter un composant à ces réseaux de communication. Elles peuvent par exemple être destinées à adapter un processeur à un réseau de communication tel que le bus *AMBA* [ARM01, MIPS02]. Ces interfaces sont mêmes parfois directement intégrées dans le processeur [ARM02].

A partir de ces standards et des méthodologies associées, des outils de conception système ont vu le jour. *Platform Express* [Men04] est un logiciel destiné à accélérer le processus d'intégration d'IP autour d'une architecture processeur. Cet outil est destiné à des architectures construites à base de bus partagé. *Platform Express* repose sur une technologie d'encapsulation d'IP permettant de saisir le système sous forme graphique. Il est possible de générer l'IP en fonction de quelques paramètres de configuration sans avoir accès au code source. Le lien entre *Platform Express* et l'outil de simulation

logiciel/matériel *Seamless* [Men03] de *Mentor Graphics* permet de vérifier le système après l'avoir saisi. On sélectionne un composant dans un catalogue de composants, puis on l'ajoute au système en utilisant un éditeur de schéma. L'outil connecte alors automatiquement le composant au bus correspondant, et ajoute au besoin des ponts pour adapter le composant au bus et des contrôleurs d'interruptions pour gérer les interruptions des processeurs.

Plusieurs autres outils d'intégration de composants autour de réseaux de communication "propriétaires" méritent également d'être cités. *SonicsStudio* est un outil commercial développé par la société *Sonics* [Sonics] pour l'intégration de composants autour de ses solutions d'interconnexions configurables [Win01]. L'approche est en partie similaire à celle de *Platform Express*. L'architecture du système est saisie avec une interface graphique. Mais l'interface des composants à connecter au réseau est restreinte au standard *OCP* et le réseau de communication est composé uniquement de bus créés par *Sonics*. Une fois le système saisi, l'outil permet de générer soit un modèle de simulation de haut niveau avec des modèles de bus fonctionnels, soit un modèle RTL en VHDL ou en Verilog.

La société *Arteris* [Arteris] a adapté cette approche au cas des réseaux sur puces. Le réseau de communication est un réseau sur puce commercialisé par *Arteris*. L'outil *NoCcompiler* permet la configuration et la création d'un réseau sur puce, ainsi que l'assemblage de composants autour du réseau généré. L'outil supporte plusieurs interfaces de composants, les interfaces des composants étant également des composants réutilisables rangés dans une bibliothèque. A la différence des outils précédents, la topologie du réseau de communication n'est pas saisie par une interface graphique, mais est générée par l'outil *NoCexplorer* qui assiste le concepteur dans l'exploration de plusieurs topologies de réseaux.

2.5.2 Génération d'interfaces avec des architectures génériques

a) L'outil ASAG :

L'outil ASAG [Lyo01, Lyo03] est un outil de génération d'interfaces de communication faisant partie du flot de conception système ROSES, développé au sein du groupe SLS du laboratoire TIMA. Il raffine des modèles abstraits de l'architecture du système en un modèle RTL. La communication est modélisée par des canaux de communication reliant les composants du système. L'outil ASAG permet la génération d'interfaces de communication entre des sous-systèmes processeurs et les canaux de communications qui seront raffinés en des connexions point à point. [Gha03] a étendu cette approche à la génération d'adaptateurs mémoires.

L'outil ASAG fait de l'assemblage de composants suivant l'architecture générique de la figure 2.12.

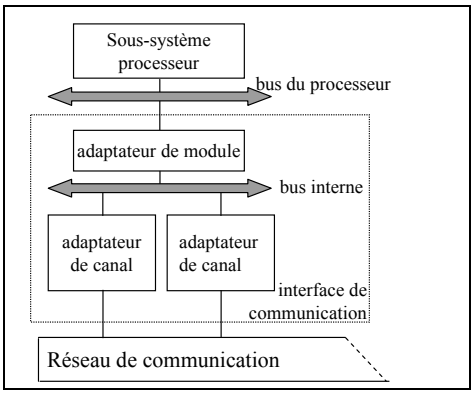


Figure 2.12 - Architecture des interfaces générées par ASAG

L'adaptateur de module sert d'interface entre un bus interne et le bus du processeur, et les adaptateurs de canaux adaptent le bus interne au réseau de communication. L'adaptateur de module est spécifique au processeur. Il s'occupe de la gestion des interruptions, du décodage d'adresses et de la communication avec les adaptateurs de canaux. Les adaptateurs de canaux sont spécifiques aux canaux de communication et communiquent avec l'adaptateur de module à travers le bus interne. Ils sont chargés d'adapter le protocole du bus interne au protocole du canal de communication.

Un adaptateur de canal est généré pour chaque canal de la spécification du système. L'outil ASAG sélectionne et configure les composants stockés dans sa librairie. Les composants sont écrits avec des macro au niveau RTL, avec un langage HDL (VHDL ou SystemC). ASAG paramètre les composants de bibliothèque en fonction de la spécification du système.

L'utilisation de bibliothèques permet d'accélérer la conception des interfaces de communication, mais le travail de conception se trouve reporté dans la création des bibliothèques. Les outils permettent de réaliser le travail de configuration des adaptateurs de canaux et de l'adaptateur de module de façon automatique mais la conception des éléments de bibliothèques reste manuelle. Si la conception est très simple et très rapide dans le cas de protocoles de communications simples, comme on en trouve dans le cas de réseaux point à point, les protocoles peuvent être beaucoup plus compliqués (ex : bus AMBA, ST bus…). L'écriture des composants en HDL peut alors être fastidieuse et source d'erreurs.

b) Autres méthodes

Cosy

Dans le cadre du projet Cosy [Bru00], une interface de communication modulaire a été réalisée. Le système est spécifié par un ensemble de tâches communicantes par des canaux bloquants, qui prennent la forme de FIFO. Ce modèle est basé sur les réseaux de Kahn. C'est donc un modèle fonctionnel du système. Quand la communication est entre deux tâches logicielles, la communication se fait par une FIFO réalisée en logiciel et située dans une mémoire. Si la communication est initiée par un processeur à destination d'un coprocesseur, la FIFO est réalisée en matériel dans l'interface de communication. Le processeur est toujours maître, il n'y a donc pas de communication du matériel vers le logiciel. Les interfaces de communication sont générées à partir de librairies, et utilisent

l'architecture générique des interfaces décrite dans [Hom01a, Hom01b]. Elles utilisent des FIFO communicant à travers le protocole VCI [Len00] avec un adaptateur réalisant le protocole de communication désiré. Pour l'instant, le réseau de communication utilisé est un PI-Bus. Les interfaces permettent la réutilisation du réseau grâce à l'interface VCI. La communication avec les IP est réalisée avec un protocole particulier via un adaptateur. Les interfaces possèdent également un module de registres de configuration, un module de registres d'états et un concentrateur d'interruptions. Les processeurs communiquent à travers une interface VCI avec les interfaces.

EASI-Tools

[Bea] propose une suite d'outils pour faciliter l'intégration et la réutilisation d'IP, appelé *EASI-Tools* ("*Embedded Application System Interfaces"*). L'objectif est de permettre un développement concurrent du logiciel, du matériel et de la documentation. La suite *EASI* génère des API logiciel de bas niveau (HAL), des interfaces de communications décrites en HDL, un modèle HDL des bus et du système (netlist), et une documentation de l'ensemble. Les API permettent d'accéder et de modifier les registres. L'outil part d'une description du système (réalisée avec une interface graphique). Les IP sont décrits dans une base de données en XML avec le format SPIRIT. Le principe de l'outil s'inscrit dans la lignée des travaux de [Hom01a, Lyo03]. Il repose sur une architecture générique et des composants de bibliothèque. Il cible les interfaces des IP, et se base sur une description des registres de l'IP. EASI-Tools suppose que les registres de configuration ou de communication sont externes au cœur de l'IP, et les génère avec la logique d'adaptation du bus. L'interface sépare la logique spécifique au bus et les registres en deux blocs communiquant à travers une interface *VCI*.

Mais aussi …

Le lecteur pourra également se référer aux travaux de [Ver96] qui génère des interfaces pour des connexions point à point et de [Rad04] qui génère une interface pour un réseau sur puce particulier.

2.5.3 Méthodes de synthèse de haut niveau pour la conception des interfaces de communication

a) Protocol Compiler

Protocol Compiler [Syn01a, Sea96] est un outil, anciennement commercialisé, issu des travaux de recherche de [Sea94]. Il permet la génération de la logique de contrôle pour des systèmes manipulant des flux de données structurées, et tout spécialement la logique de contrôle de protocoles de communication où les données sont organisées en trames, paquets ou cellules.

La spécification d'entrée se présente sous une forme graphique. Cette spécification du contrôleur est proche de celle du format des données à traiter avec les actions de contrôle nécessaires. Le concepteur ne décrit pas le flot d'opérations en termes d'états explicites ou de variables de contrôle. La

spécification définit hiérarchiquement des trames et des opérations sur les données associées. Les opérations sur les données et les ports d'entrée-sortie sont spécifiées par des actions.

Les trames sont hiérarchiques et peuvent être composées de trames plus simples. A chaque trame peut être attachées une ou plusieurs actions. Une trame hiérarchique étant définie par une séquence de trames, cette séquence va déterminer l'ordre des actions associées aux trames. Des expressions booléennes peuvent être attachées aux trames. L'action est alors exécutée si l'expression est validée. Il est ainsi possible de reconnaître des motifs de données spécifiques en entrée ou bien de décrire le comportement cycle par cycle. Les trames définissent donc quand on exécute les actions.

Les actions peuvent être choisies parmi un petit ensemble d'actions prédéfinies (tel qu'assigner une valeur à des variables, mettre à 0 ou 1 un port, incrémenter ou décrémenter un compteur) ou bien définies par utilisateur. Dans le cas d'une action définie par l'utilisateur, ce dernier doit alors écrire l'action associée avec un langage HDL.

Protocol Compiler permet la synthèse d'une machine d'états finis à partir de la spécification d'entrée. Cette machine d'états finis est alors optimisée suivant la méthode exposée dans [Sea98]. Le code HDL (VHDL ou Verilog) est alors généré pour obtenir le circuit du contrôleur, dont le bloc diagramme est représenté sur la figure 2.13.

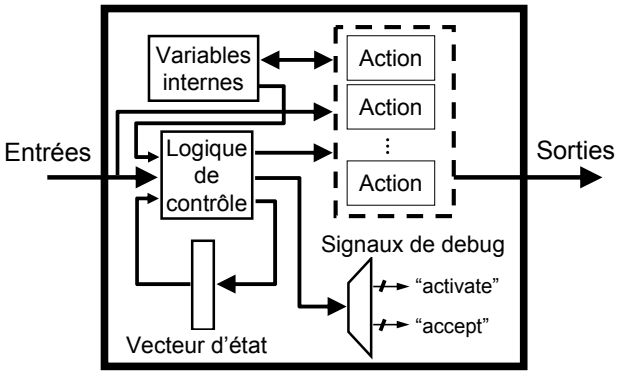


Figure 2.13 - Bloc diagramme du matériel généré

La spécification utilisée par *Protocol Compiler* ne requiert pas une description explicite des états et des transitions. Son plus haut niveau d'abstraction vise à la rendre plus proche des spécifications des systèmes ciblés par la méthode, et permet d'avoir une spécification beaucoup plus concise qu'avec une machine d'états finis. L'avantage de ce modèle de haut niveau par rapport à une FSM est qu'il facilite alors les modifications, la détection et la correction des erreurs, et réduit le temps de conception.

b) ComSyn : Utilisation du langage Program

[Öbe01, Öbe99] a proposé une méthode apparentée à [Syn01a] pour la génération de contrôleurs de communication, qui repose également sur la reconnaissance de motifs de données en entrée. Par contre, la spécification n'est pas hiérarchique mais écrit dans un langage appelé *ProGram* développé pour ce problème. La philosophie de la méthode est qu'une machine d'états finis s'apparente à la

grammaire d'un langage ; une grammaire étant interprétée comme une manière générale de définir des motifs. Une machine d'états finis devant elle reconnaître des bits et leurs séquences, [Öbe01] montre qu'une grammaire régulière peut être utilisée pour décrire une machine d'états finis. *ProGram* définit donc la “grammaire d'un protocole”, et permet une description de haut niveau d’un protocole de communication.

Il existe 5 sections dans *ProGram* pour définir les ports d’entrées/sorties et les signaux, des constantes, les mémoires pour stocker des variables, les fonctions et macro, les règles de grammaire décrivant les transitions. Les règles de grammaire sont les séquences de bits valides que l’on doit reconnaître en entrée. A chaque séquence est associée une séquence en sortie correspondante. Les protocoles de communication sont décrits indépendamment de la taille des ports, ce qui facilite la réutilisation du code et permet de rapidement étudier l’influence de la taille des ports sur les performances.

Un compilateur sert à synthétiser une machine d'états finis à partir d'une spécification écrite avec le langage *ProGram*. Les analyseurs syntaxiques des compilateurs reconnaissent des motifs dans un programme informatique. Une machine d'états finis en charge du contrôle d’un protocole fait plus ou moins la même chose qu'un analyseur syntaxique. Elle reconnaît des motifs de mots de données en entrée. La conception d'une machine d'états finis à partir de *ProGram* est donc conceptuellement très proche de la conception d'un analyseur syntaxique pour un compilateur, qui peut être fait avec des outils tel que GNU Yacc. La figure 2.14 montre l'architecture du matériel généré par *ProGram* au format VHDL.

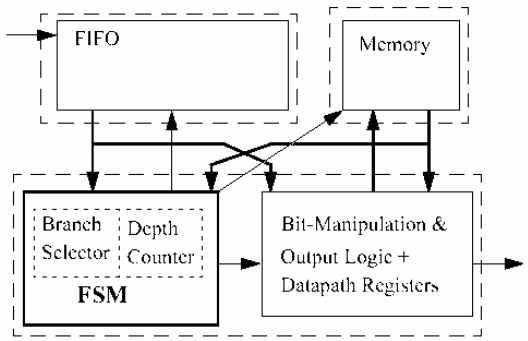


Figure 2.14 - Architecture du matériel généré par ProGram [Öbe99]

[O’Ni01] définit une méthodologie pour réaliser les communications dans les SoC basée sur le langage *ProGram*. La figure 2.15 représente l'architecture type des systèmes ciblés. Une interface de bus sert d’adaptateur entre le bus du processeur et les ports d'un composant matériel. Le pilote du composant peut être réalisé soit tout en logiciel, soit avec une réalisation mixte logiciel/matériel, où un DMA s'occupe d'une partie du contrôle de la communication.

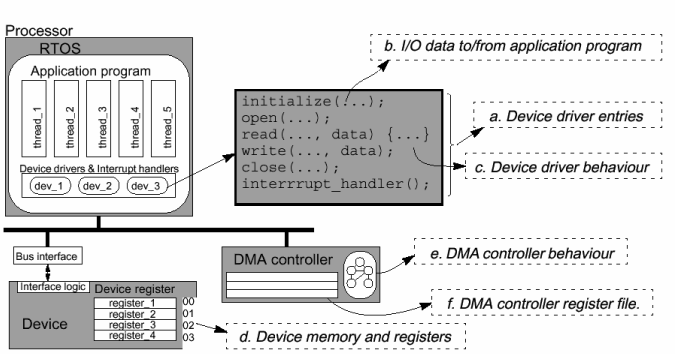


Figure 2.15 - Architecture cible [O'Ni01]

Le concepteur modélise en *ProGram* l'interface de bus et le pilote. Afin de permettre la synthèse des pilotes, [O'Ni98, O'Ni01] a travaillé sur la génération de code C à partir de la machine d'états finis extraite par le compilateur et a étendu *ProGram* pour gérer certains concepts propres au logiciel tels que les interruptions. Une partie d'un pilote peut être réalisée en matériel dans un contrôleur DMA spécifique à l'application afin d'améliorer les performances. Le contrôleur DMA est en partie généré à partir de la description *ProGram*. L'autre partie qui est dépendante du bus du processeur est contenue dans une bibliothèque.

c) Adaptation automatique

[Nar95] et [Pas98] ont proposé des méthodes pour l'adaptation automatique de deux protocoles mais de telles méthodes ne permettent de générer que des interfaces de communications relativement basiques. [Shi02] utilise la méthode de [Nar95] pour générer des interfaces à base de FIFO mais dont l'architecture est figée. [Zit02] a travaillé sur la communication par mémoire partagée dans les systèmes multiprocesseurs en utilisant la méthode de [Nar95] pour l'adaptation des processeurs au bus, mais la synchronisation entre les différentes tâches applicatives est réalisée entièrement de façon logicielle.

d) Autres méthodes

Utilisation de réseaux de Petri

[Lin94] écrit la spécification de l'interface avec un langage de haut niveau. On y définit des ports, des canaux de communications et leurs protocoles de communication, et le comportement de l'interface. Cette spécification est alors transformée en un dérivé des réseaux de Petri, où les états correspondent à des entrées-sorties sur des canaux de communications, et à des opérations sur les données. Le réseau de Petri est alors raffiné en une machine d'états finis. On alloue pour cela des blocs de traitement de données, avec lesquels on communique avec un protocole standard ("handshake" à 2 ou 4 phases). Pour communiquer avec l'extérieur, on adapte automatiquement le protocole externe avec un des protocoles internes. Pour cela, on a une librairie de protocoles décrits à haut niveau (~réseaux de Petri). Une seule description du protocole sert pour l'émetteur et le receveur, les

transitions devenant des états et les états des transitions pour passer de l'un à l'autre. Les protocoles sont annotés pour indiquer la présence des données et des adresses. Un compilateur de circuits asynchrones génère l'interface au niveau RTL. Les FIFO sont considérées comme des modules externes.

Dans [Via98], l'interface est également décrite avec un langage spécifique et les protocoles sont modélisés avec des réseaux de Petri. La méthode de synthèse permet en revanche d'obtenir des circuits synchrones.

SV

[Sie02] génère un contrôleur de communication matériel à partir d'une spécification du protocole de communication, écrite avec une extension de SystemC prénommée SV. Le protocole est décrit avec un modèle unique à partir duquel on génère le contrôleur à la fois de l'émetteur et du récepteur. A partir de la spécification, on obtient des machines d'états finis décrites en SystemC que l'on synthétise.

GAUT

GAUT [Bag98] est un outil de synthèse comportementale dédié pour les applications de traitement du signal et de l'image. Il est basé sur une architecture du système composé d'un DSP et un ASIC relié par une unité de communication. L'ASIC est généré à partir d'une description de haut niveau avec le logiciel GAUT. L'interface de communication entre le DSP et l'ASIC est ensuite générée à partir de la description du matériel. L'architecture des interfaces est figée. Elle est composée de FIFO, d'un adaptateur entre les entrées sorties du processeur et les FIFO, d'un adaptateur entre l'ASIC et les FIFO, et d'un contrôleur. Le contrôleur s'occupe de l'ordonnancement des communications. Il est généré à partir d'une description des séquences de communication et des contraintes de temps.

2.5.4 Choix d'une méthode de conception des interfaces de communication

Un certain nombre de travaux [Syn01a, Sie02] ont étudié la génération de la logique de contrôle de protocoles de communication mais le besoin de réutiliser les composants (IP et processeur) rend ces travaux mal adaptés au problème de la synthèse au niveau système, car ils ne traitent pas du problème de l'adaptation de protocoles. A cette fin, des standards de protocole tels que OCP [OCP03] ont été proposés. Malgré tout, aucun standard ne s'est réellement imposé et leur utilisation pour les processeurs peut parfois augmenter les temps de communication [Cyr01]. [Nar95, Pas98] ont traité de l'adaptation de protocoles, mais la réalisation d'interfaces de communication avec un comportement plus compliqué, tel que l'utilisation de FIFO, impose alors de figer l'architecture [Shi02]. Les travaux de [Lin94] permettent de modéliser des comportements plus compliqués.

Mais cela ne règle pas le problème de l'utilisation du logiciel qui impose l'utilisation de modèles de haut niveau comme les réseaux de Kahn. [Hom01a] fixe le modèle de communication tandis que

[Lyo03] reste limité par la disponibilité des bibliothèques. [O'Ni01] gère le problème de l'hétérogénéité logiciel/matériel et permet une accélération matérielle du pilote avec l'utilisation d'un DMA, mais la modélisation reste manuelle.

Notre objectif est la réalisation d'interfaces de communication telles que celles existantes dans les réseaux et adaptées aux cas des NoC par [Rad04]. On ne considère plus l'interface juste comme un adaptateur de bus [ARM01] mais plutôt comme les interface utilisées dans les réseaux d'ordinateurs [Cul99b]. L'interface devra être conçue conjointement avec le pilote de communication de manière à optimiser les communications. Nous voulons une méthode de conception flexible permettant de choisir quelles fonctionnalités réaliser dans le pilote ou dans l'interface de communication. Du fait de leur haut degré d'automatisation, on considère que les méthodes basées sur l'assemblage de composants de bibliothèques sont les plus intéressantes, mais pour répondre à nos objectifs une telle méthode ne doit pas être basée sur une architecture générique.

2.6 Conclusion

La communication est un élément clé de la conception des systèmes monopuces. Des unités de calcul de plus en plus élaborées (destinées à des applications elles aussi plus élaborées) vont imposer des contraintes de performances de plus en plus sévères sur les réseaux de communications. Les contraintes physiques des interconnexions qui s'ajoutent à ces contraintes entraînent l'utilisation de réseaux de plus en plus compliqués. Cette évolution influe évidement sur la conception des interfaces de communication. Dans le même temps, les impératifs de conception logicielle requièrent d'offrir une abstraction du matériel au programme applicatif. Cette abstraction se paye elle aussi par une complexité croissante des interfaces de communications. Le problème de la conception des interfaces de communication est d'autant plus difficile que plusieurs solutions de réalisation logiciel/matériel sont possibles et qu'il convient de trouver un compromis entre ce que l'on veut réaliser en logiciel et ce que l'on veut réaliser en matériel. Trois grands enseignements se dégagent de ce chapitre et ont servi à guider le reste du travail présenté dans ce manuscrit. Le premier est la prise en compte de la conception du logiciel au moment de la conception des interfaces de communication, ce qui implique d'utiliser une même approche pour la conception des pilotes et des interfaces de communication. Le deuxième est le besoin d'adapter les interfaces de communications au système, ce qui impose d'avoir une méthode de conception flexible. La troisième est la nécessité d'avoir une méthode de haut niveau d'abstraction pour supporter la complexité croissante des interfaces.

Chapitre 3

Modélisation et raffinement des communications dans le contexte des SoC

Sommaire

3.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, la communication est un élément primordial dans un système monopuce au même titre que les algorithmes de traitement des données, ou que le choix d'un processeur ou d'un IP. Elle doit donc être étudiée avec d'autant plus de soin qu'elle requiert des réseaux de communication complexes.

La nécessité d'optimiser les communications d'un système impose de concevoir avec attention les interfaces de communication. Ceci est d'autant plus difficile que le système doit généralement supporter des applications logicielles utilisant des modèles de programmation de haut niveau simplifiant le développement de ces applications. La complexité des protocoles mis en jeu complique alors la conception. Il est donc nécessaire de développer des méthodes et des outils permettant de transformer des modèles de haut niveau du protocole en une réalisation logiciel/matériel des communications. Cette tâche ardue l'est d'autant plus que différents choix de réalisation logiciel/matériel sont envisageables. La réalisation de certaines fonctionnalités en matériel peut offrir des gains en performance.

Ce chapitre présente un flot de génération des interfaces logiciel/matériel et situe la génération des interfaces de communication à l'intérieur de ce flot. Les modèles donnés en entrée du flot de génération des interfaces logiciel/matériel sont ceux de l'architecture du système et de la description des services de communications utilisées par l'application.

3.2 Préliminaire : Introduction aux méthodes à base de graphes de dépendances de services

3.2.1 Les graphes de dépendances de services

Le chapitre 2 a mis en valeur l'intérêt des méthodes de génération d'interfaces de communication basées sur l'assemblage de composants de bibliothèques. Cet assemblage se fait suivant des architectures génériques. Cela pose des problèmes de flexibilité, et ne permet pas de résoudre les problèmes de réalisation mixte logiciel/matériel des communications. Nous souhaitons donc une méthode permettant l'assemblage de composants suivant différentes architectures et permettant de gérer des composants aussi bien matériel que logiciel. Cette idée a donc amené à étudier les méthodes de synthèse logicielle existantes.

Un certain nombre de méthodes pour la réutilisation, la configuration et l'assemblage des composants logiciels se basent sur des graphes de dépendance de services. Les graphes de dépendances de services sont des modèles couramment utilisés dans le génie logiciel, aussi bien pour la gestion et la programmation de systèmes distribués [Che03, Ens01], que la configuration de systèmes d'exploitations [Bök00, Gau01] ou la programmation de sous-systèmes de communications [Zit93].

La notion de service a déjà été introduite dans le chapitre 2 pour la description des modèles de communication en couches, tel que le modèle de référence OSI. L'approche proposée par cette thèse repose sur la modélisation des communications avec un graphe de dépendance de services. L'idée de base du flot développé au sein de l'équipe SLS du laboratoire TIMA est d'étendre les méthodes de génération de graphes de dépendance de services à l'ensemble du flot de conception des interfaces logiciel/matériel, alors qu'actuellement ces méthodes ne sont utilisées que pour la synthèse logicielle. Les graphes de dépendance de services modélisent des composants logiciels dans les travaux existants. Les travaux menés pendant cette thèse étendent l'utilisation de ces graphes à la modélisation de composants matériels afin d'utiliser un unique modèle pour l'interface et le pilote.

3.2.2 Principe des méthodes de génération d'un graphe de dépendance de services

Des travaux ont déjà traité de la génération de graphes de dépendance de services dans le génie logiciel. Une des applications des graphes de dépendances de services est la construction de systèmes d'exploitation sur mesure pour une application et une architecture donnée [Bök00, Gau01]. Les différents blocs du système d'exploitation sont alors contenus dans une bibliothèque qui est modélisé par un graphe de dépendance de services (SDG de l'anglais : "Service Dependency Graph"). Ce graphe permet de spécialiser et configurer le système d'exploitation en ne choisissant que les blocs nécessaires pour fournir les services désirés. Par rapport à une configuration manuelle, ces méthodes permettent une spécialisation très fine tout en étant automatique.

a) L'outil ASOG

Dans [Gau01], la bibliothèque est structurée à l'aide de trois concepts de base : l'élément, le service et l'implémentation. Un élément est une partie du système d'exploitation, et un service est une fonctionnalité du système d'exploitation. Chaque élément requiert et fournit des services aux autres éléments. La dépendance entre les services permet de lier les différents éléments de la bibliothèque. Il existe donc une relation de dépendance indirecte entre éléments, même si un élément ne requiert jamais un autre élément en particulier. Il requiert un service, et à priori tout élément fournissant ce service peut convenir.

Chaque élément possède un ensemble d'implémentations. L'implémentation est une réalisation particulière de l'élément pour un ensemble de processeurs et de périphériques donnés. En pratique, une implémentation se compose donc d'un ensemble de fichiers sources, écrits dans un langage approprié (assembleur ou C).

L'ensemble des différents éléments de la bibliothèque constitue un ensemble de blocs de base pour la construction d'un système d'exploitation. Le code final du système d'exploitation peut être obtenu par l'assemblage d'implémentations paramétrées, la structure du système d'exploitation étant elle modélisée par un ensemble d'éléments liés. On peut donc distinguer deux représentations dans la

bibliothèque (figure 3.1). La représentation comportementale correspond aux composants logiciels, et à leurs codes sous la forme de macro. Une représentation relationnelle liant ces composants a pour objectif de permettre de retrouver et identifier les divers éléments, grâce à leur représentation sous forme d'éléments et de services.

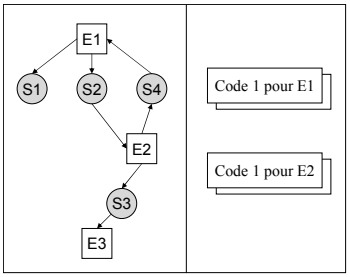


Figure 3.1 - La bibliothèque de système d'exploitation de l'outil ASOG [Gau01]

La spécialisation du système d'exploitation modélisé dans la bibliothèque consiste à choisir uniquement les éléments nécessaires pour la construction d'un système d'exploitation spécifique à une architecture et une application donnée. Cette spécialisation se fait à l'aide du graphe de dépendance de services constitué par les éléments et services de la bibliothèque. Sur la figure 3.2, un arc orienté d'un élément vers un service correspond à un service requis par l'élément, et inversement, un arc orienté d'un service vers un élément correspond à un service fourni.

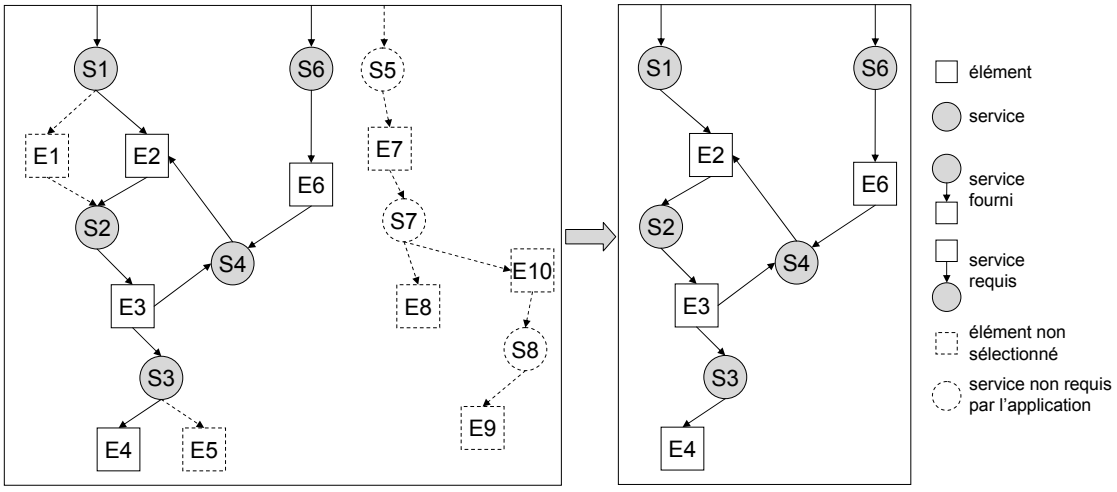


Figure 3.2 - Génération d'un SDG

L'outil de spécialisation prend en entrée une description de haut niveau de l'application et de l'architecture. La construction du graphe se fait par sélection de tous les éléments et services valides pour l'architecture en dépendance directe ou indirecte avec les services initiaux, correspondant aux fonctions d'API utilisées par les tâches du programme applicatif. Ces services constituent la racine du graphe. Un élément est considéré comme valide s'il a une implémentation compatible avec l'architecture matérielle, notamment avec le type du processeur. L'outil construit la liste des éléments

nécessaires et suffisants pour fournir tous les services racines, et choisit donc entre plusieurs éléments proposant les mêmes services. A cette fin, des médias sont indiqués dans la spécification d'entrée. Les médias permettent de spécifier des choix de réalisation pour les communications entre les tâches d'un même nœud, ou avec l'extérieur en fonction du réseau et du périphérique utilisé. Un élément est valide s'il n'utilise pas de médias non présents dans la spécification d'entrée. Les médias indiquent donc les périphériques présents dans l'architecture et guident le choix des pilotes de périphériques correspondants.

Si la méthode ne permet pas de faire un choix entre plusieurs éléments fournissant le même service, l'utilisateur choisit l'élément à conserver. De plus, dans le cas où un composant sélectionné fournit un service qui n'est requis par aucun autre élément sélectionné, une partie du code de l'implémentation de l'élément liée à ce service ne sera pas incluse afin de ne mettre que le code nécessaire.

b) L'outil TEReCS [Bök00]

[Bök00] utilise un SDG pour la synthèse logicielle pour des systèmes temps réels distribués. Des composants logiciels extraits de DREAMS sont assemblés pour obtenir le code final. DREAMS est un ensemble de constructions de systèmes d'exploitations basé sur une bibliothèque paramétrable. DREAMS est un système orienté objet, où la configuration n'est pas seulement appliquée aux classes ou aux objets mais aussi au niveau de l'interface. DREAMS a servi de base à [Bök00] pour la synthèse du système d'exploitation. L'outil TEReCS sert à la configuration et à la spécialisation de DREAMS. Pour TEReCS, les plus petits fragments de logiciel pouvant être configurés sont modélisés par des services. Chaque service dépend d'autres services. L'ensemble des services et des dépendances forme un graphe de dépendance de services (figure 3.3).

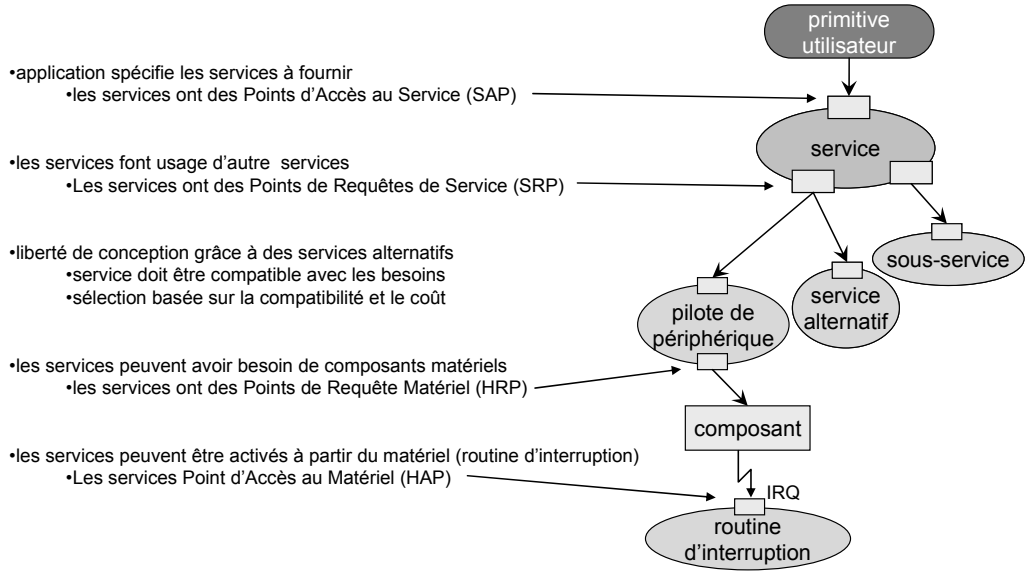


Figure 3.3 - Le graphe de dépendance de services utilisé par l'outil TEReCS [Bök00]

Les services sont utilisés via des Points d'Accès au Service (SAP pour “Service Access Point”), et utilisent d'autres services par des Points de Requêtes de Services (SRP pour “Service Request Point”). Le graphe de dépendance de services gère les codes des différents composants et leur dépendance. Ce graphe sert de modèle de configuration à l'outil TEReCS. DREAMS a donc été modélisé avec un graphe de dépendance de services pour être configuré avec TEReCS. Chaque classe et API de DREAMS est donc représenté par un service.

La configuration et la spécialisation se font là aussi automatiquement à partir d'une description des services requis par l'application. Ces services sont modélisés par des primitives utilisateurs, sorte d'API. Les primitives utilisateurs sont connectées aux SAP des services correspondant dans le graphe des services. A partir de ces services, l'outil doit sélectionner l'ensemble minimal des services requis depuis l'application à partir des relations de dépendances alternatives. Le sous-arbre complet de services partant des primitives utilisateurs, les racines du graphe, suivi de tous les services attachés aux SRP doit être sélectionné.

L'outil de configuration est capable de sélectionner des services entre deux primitives utilisateur, tel qu'une primitive de transfert et de réception de deux processus différents par exemple. Un “Graphe de Ressource Universel” modélise aussi les entités matérielles (processeur, périphérique et media de communication), permettant de tenir compte des communications externes (entre deux applications sur deux noeuds différents par exemple).

Différents mécanismes ont été utilisés pour choisir entre les différentes alternatives existantes dans les dépendances de services. Ces mécanismes sont appliqués dans l'ordre suivant :

- ➔ la modélisation du matériel permet de choisir uniquement les services supportés par le matériel (exemple : le choix entre différentes alternatives de pilotes de périphériques, modélisés par différentes services, est conditionné par les différents périphériques effectivement présents)
- ➔ des services déjà sélectionnés peuvent inhiber la sélection d'autres services
- ➔ il est possible de restreindre le nombre d'utilisations d'un service par d'autres services
- ➔ des priorités sont assignées aux services équivalents pour fixer les services à choisir en premier lieu
- ➔ en cas de services avec la même priorité, le service le plus fréquemment requis par d'autres services est choisi
- ➔ s'il existe encore des services équivalents, une fonction coût permet de choisir le service impliquant la taille de code la plus faible

3.2.3 Limites des modèles en couches et configuration de protocole

a) Modèle de communication

Structurer l'abstraction en couches est un bon moyen pour gérer la complexité des communications (chapitre 2.4), mais garder la même structure pour la réalisation logiciel/matériel des communications peut constituer une limitation en termes de performance et de flexibilité [Zit93].

Sur la voie du circuit final, il peut être nécessaire de fusionner certaines couches pour des raisons d'optimisation lors du développement logiciel/matériel [Sgr01, Mil04]. Un modèle en couches se révèle donc mal adapté au problème de la conception car sa structure se révèle éloignée de la réalisation finale. [Zit93] a proposé un modèle offrant les facilités d'abstraction et de modélisation des modèles en couches, plus facilement utilisable sur des architectures multiprocesseurs. Dans les réseaux, un sous-système de communication est parfois dédié uniquement à l'exécution du logiciel contrôlant les communications. Une plateforme multiprocesseur est une architecture très intéressante pour un tel sous-système grâce à ses performances.

Le modèle de communication de [Zit93] repose sur l'utilisation de "fonctions de protocoles" au lieu de couches de protocoles comme unités configurables (figure 3.4).

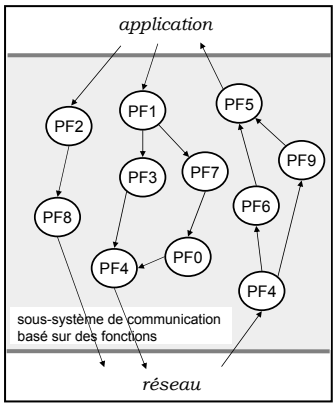


Figure 3.4 - Un modèle de communication basé sur des fonctions [Zit93]

La plus grande flexibilité du modèle de communication proposé doit permettre au sous-système de communication de supporter une interface d'application, où l'application peut requérir du sous-système de communication des services adaptés à ses besoins individuels. Ces exigences particulières des services de communications en performances et en fonctionnalités peuvent être formulées avec des paramètres de services ou des fonctions de protocoles. Les paramètres définis sont aussi bien quantitatifs (débit, latence, temps de réponse, taux de transfert, "jitter", seuil de corruption de données, seuil de perte de données) que qualitatif (contrôle de session, contrôle de flux, tolérance d'erreurs). Pour des applications sans exigences spéciales, quatre classes de services ont été prédéfinies déchargeant l'application de la spécification en détail des besoins des services, qui impose de manier un grand nombre de paramètres. La sélection entre ces 4 classes de services dépend juste du fait que l'application soit temps réel ou non, et que la transmission doive être fiable ou non.

b) Outil de configuration

Le sous-système de communication défini est composé d'un ensemble de machines de protocoles qui sont dynamiquement créées, modifiées ou effacées. Une machine de protocoles est un ensemble de fonctions de protocoles sélectionnées et combinées. Un configurateur de protocoles les configure automatiquement en fonction de l'application. La description de la machine de protocoles obtenue est

alors passée à un configurateur de code, qui génère le code pour une plateforme matérielle qui peut être éventuellement multiprocesseur.

Une machine de protocoles par défaut est définie pour chaque classe de services prédéfinis. Celle-ci peut être configurée pour être optimisée aux besoins spécifiques de l'application. On peut distinguer cinq types de fonctions :

- Les fonctions du noyau (“kernel function”) qui sont automatiquement sélectionnées pour toutes les machines de protocoles
- Les fonctions requises qui sont sélectionnées pour une certaine classe de service
- Les fonctions requises par l'application qui sont sélectionnées par une requête explicite de l'application ou bien en fonction de paramètres qualitatifs
- Les fonctions requises par le réseau, leur sélection dépend des caractéristiques du réseau et est décidée par le sous-système de communication.
- Les fonctions sans rapport avec une certaine classe de services et qui ne peuvent être sélectionnées pour cette classe

Cette distinction sert à guider le choix des fonctions de protocoles. On distingue deux niveaux dans ce choix (figure 3.5). Certains éléments sont directement reliés à certains critères de services (premier niveau de fonctions de protocoles), et sont choisis en fonction de ces critères. Ces éléments peuvent être dépendants d'autres éléments, qui sont alors sélectionnés pour supporter le premier niveau de fonctions.

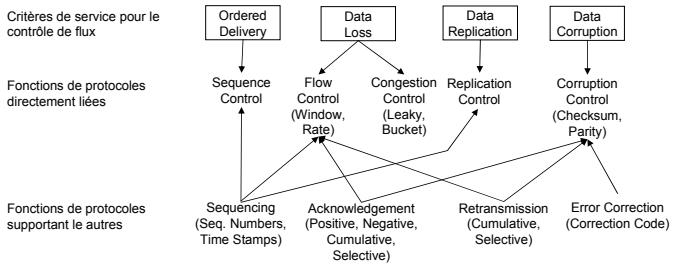


Figure 3.5 - La sélection des fonctions de protocoles [Zit93]

3.3 Modélisation du système

3.3.1 Le concept d’interface logiciel/matériel

Comme nous l’avons vu précédemment, un point fondamental de la complexité des systèmes monopuces est l’utilisation de processeurs dans les circuits intégrés. Le développement d’un flot de conception système nécessite donc la maîtrise de l’hétérogénéité logiciel/matériel et la modélisation des interactions entre le logiciel et le matériel. On parle d’interface logiciel/matériel pour qualifier les liens entre le logiciel et le matériel. Pour la définir, il est nécessaire de comprendre comment s’organise un sous-système processeur. Un sous-système processeur permet l’exécution du logiciel.

Avec l’évolution des systèmes monopuces, la complexité du logiciel a également augmenté. Afin d’en gérer la complexité, on le structure traditionnellement en couches, comme le montre la figure 3.6.

Le programme applicatif se compose d'un ensemble de tâches communicantes, réalisant le comportement de l'application tel que décrit dans la spécification fonctionnelle. Les applications s'exécutent sur un système d'exploitations (SE). Celui-ci est responsable de l'ordonnancement des tâches, et fournit certaines fonctionnalités propres au système. La couche d'abstraction du matériel, le HAL (de l'anglais "Hardware Abstraction Layer"), est une couche d'abstraction de l'architecture matérielle. Elle permet le portage du SE et du programme applicatif sur différentes machines.

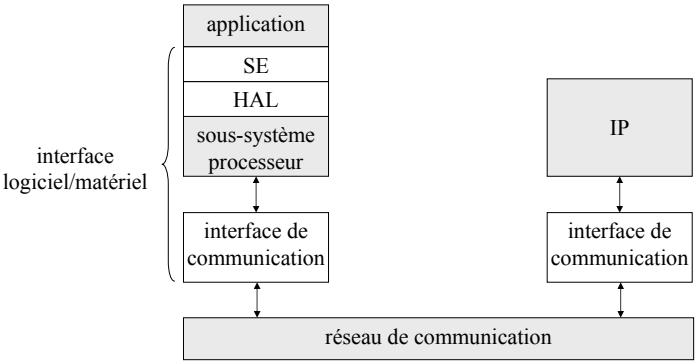


Figure 3.6 - Les couches matérielles/logicielles

Le logiciel s'exécute sur un sous-système processeur communiquant avec le réseau à travers l'interface de communication. Un sous-système processeur est défini comme un processeur auquel peut être adjoint des périphériques et des mémoires accessibles localement par le logiciel (sans passer par le réseau de communication). Ces périphériques peuvent être des coprocesseurs matériels ou des périphériques nécessaires au fonctionnement du processeur (exemple : contrôleur d'interruptions, contrôleur mémoire). L'interface logiciel/matériel est composée du SE, du HAL, du sous-système processeur et de l'interface de communication. L'application communique avec le reste du système à travers le réseau de communication. Pour cela, elle appelle des API tandis que l'interface avec le réseau se fait à travers un ensemble de ports physiques. L'interface logiciel/matériel permet donc d'interfacer une application logicielle avec un réseau de communication matériel.

3.3.2 Spécification du système : l'architecture virtuelle

Dans notre méthodologie, le système est modélisé par une architecture virtuelle [Ces02]. Une architecture virtuelle est définie comme étant une architecture abstraite composée d'un ensemble de modules virtuels connectés par un réseau de communication abstrait (figure 3.7).

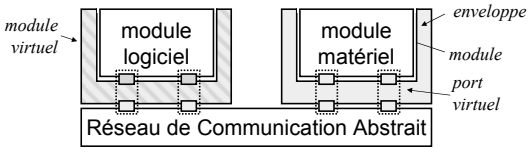


Figure 3.7 - Une architecture virtuelle

Le réseau de communication abstrait est un modèle du réseau de communication défini au niveau d'abstraction transactionnel [Hav02, Don04]. Un module virtuel est composé d'un module et de son enveloppe. Un module représente un nœud de calcul, et peut être de deux types : logiciel ou matériel. L'enveloppe représente l'adaptation du module au réseau de communication. L'enveloppe est composée de ports virtuels qui offrent une abstraction des communications. Un module logiciel est un ensemble de tâches faisant partie de l'application. Il représente donc une abstraction du sous-système processeur et du logiciel système (code de démarrage, système d'exploitation, pilotes des périphériques du sous-systèmes). Un module matériel est un modèle de haut niveau d'abstraction d'un composant matériel spécifique (IP). L'enveloppe d'un module logiciel est utilisée pour abstraire les pilotes de communication, le processeur et l'interface de communication. L'enveloppe d'un module matériel abstrait l'interface de communication. L'enveloppe offre donc une abstraction des communications au module.

L'intérêt du modèle se situe dans l'utilisation d'une enveloppe pour "empaqueter" le module. L'enveloppe assure le découplage de la communication et du calcul, pour répondre aux besoins argumentés dans le chapitre 1.3.2. En effet, le concepteur peut assembler des composants même s'ils sont incompatibles en termes de protocoles, d'interfaces physiques ou de niveaux d'abstractions. Le choix des ressources de calcul peut donc être fait indépendamment des ressources de communication.

3.3.3 Modélisation de l'interface logiciel/matériel par un graphe de dépendance de services

a) Le concept de service de communication

Les ports virtuels possèdent des ports internes et des ports externes. Les ports internes sont connectés au module et les ports externes au réseau de communication (figure 3.8). Les tâches de l'application, qui constituent un module logiciel, utilisent des API pour leur communication. Ces API correspondent aux ports internes de l'enveloppe avec le module. Le réseau de communication est lui modélisé par un modèle transactionnel, qui est un modèle abstrait du réseau. L'enveloppe accède à ce réseau par ses ports externes.

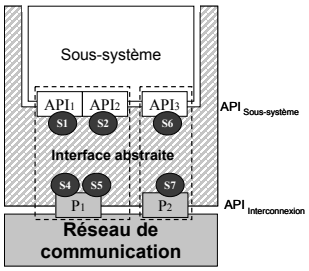


Figure 3.8 - Interfaces de l'enveloppe

Le programme applicatif cherche à utiliser une fonctionnalité de communication en utilisant ses API. Ces besoins de communications peuvent être modélisés par des services de communication. A

chaque API correspond un service de communication associé, que l'application requiert en utilisant l'API. Ce service est fourni par l'enveloppe, qui pour cela utilise les ressources de communication du réseau. Ces ressources fournies par le réseau sont également modélisées par des services. La spécification des ports virtuels décrit les services que les ports virtuels fournissent au programme applicatif pour communiquer avec les autres modules par l'intermédiaire du réseau de communication, en utilisant les services de ce réseau. Le protocole de communication décrit la façon dont les ports virtuels fournissent les services requis en utilisant les services fournis.

b) Le graphe de dépendance de services

Les ports virtuels sont spécifiés avec un modèle des communications similaire aux modèles utilisés dans le domaine des réseaux d'ordinateurs [Tan96, Zit93]. A partir de la spécification des services de communication, le concepteur doit écrire la spécification de l'enveloppe. L'élément de base du modèle est l'élément de protocole (figure 3.9). La composition d'éléments de protocole pour la modélisation des communications a été proposée dans [Zit93]. Le modèle proposé ici est une extension du modèle de [Zit93].

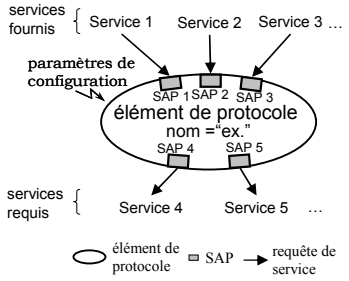


Figure 3.9 - L'élément de protocole

Un élément de protocole est un composant fonctionnel. Il peut donc être implémenté aussi bien en matériel qu'en logiciel (dans [Zit93], un élément de protocole représentait uniquement un composant logiciel). Chaque élément de protocole remplit un certain nombre de fonctionnalités qu'il propose sous la forme de services aux autres éléments du système. Il peut lui-même avoir besoin d'autres éléments pour remplir ces fonctionnalités, il requiert alors un certain nombre de services du reste du système. Chaque élément de protocole requiert et fournit des services aux autres éléments de protocole. Un service est un ensemble de fonctionnalités fournies par un élément de protocole à un autre élément de protocole. Une flèche sur la figure 3.9 représente une requête de service.

Les services des éléments de protocole sont utilisés par l'intermédiaire de Points d'Accès aux Services, aussi appelé SAP (de l'anglais : "Service Access Point"). Les éléments de protocole ont des SAP. A chaque SAP est associée une liste de services requis ou fournis. Afin d'avoir des éléments flexibles, chaque élément de protocole a quelques paramètres de configuration. Un élément de protocole se définit formellement par son nom, une liste de paramètres de configurations, et une liste de SAP. Un SAP est lui défini formellement par son nom, le nom des services, un type (fournis ou

requis). Le concept de SAP, qui a été défini dans le modèle OSI, n'était pas présent dans le modèle de [Zit93]. Il a été introduit au cours des travaux menés sur la génération automatique des interfaces pendant cette thèse pour des raisons qui sont explicités plus loin.

La spécification des ports virtuels est composée d'éléments de protocole (figure 3.10), décrivant les différentes fonctionnalités que l'interface doit fournir pour répondre aux besoins de communication. Pour écrire la spécification des ports virtuels, les éléments de protocole à utiliser sont choisis et assemblés de façon à définir le comportement global de l'interface par composition de ces fonctionnalités de base. La spécification se fait donc en liant les différents éléments de protocole. Chaque élément de protocole fournissant un service requis convient à la réalisation de ce service. La spécification des ports virtuels décrit donc quel élément de protocole fournit chaque service requis. Le lien entre un service requis et fourni est établi en liant les points d'accès aux services. L'ensemble des éléments de protocole réalise un ensemble d'opérations définissant le protocole de communication.

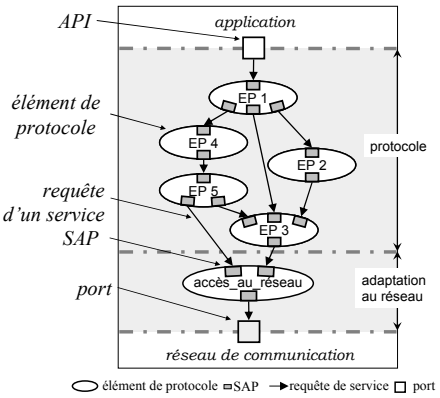


Figure 3.10 – Spécification d'un port virtuel

c) Exemple

La figure 3.11 représente un exemple de spécification d'un port virtuel. L'application envoie des messages à un autre module sur un bus AMBA AHB. Les messages sont segmentés en paquets. Les paquets sont ensuite stockés dans une mémoire tampon puis envoyés sur le bus.

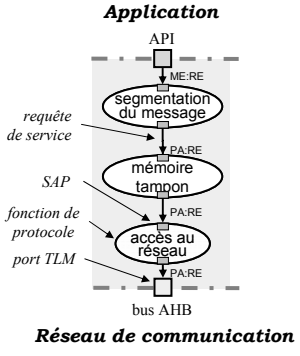


Figure 3.11 - La spécification d'un port virtuel

L'application accède au service "envoyer un message" (ME:RE sur la figure) par l'intermédiaire d'une API. Les trois étapes de la communication sont modélisées par trois éléments de protocole.

L'élément de protocole "*segmentation du message*" segmente le message en paquets qui sont ensuite transmis par appel du service "envoyer un paquet" (PA:RE). L'élément de protocole "mémoire tampon" fournit ce service mais ne fait que mémoriser le paquet (données + adresse destination) et appelle également le service "envoyer un paquet" (PA:RE). Ce service est fourni par l'élément de protocole "accès au réseau" qui accède au bus AMBA par l'intermédiaire d'un port maître. Cet élément de protocole utilise le service "envoyer un paquet" (PA:RE) du bus qui représente la possibilité de faire des transferts en rafale sur le bus. L'élément de protocole *"accès au réseau"* est responsable du transfert physique des données sur le réseau de communication.

Comme l'élément de protocole *"mémoire tampon"* offre et requiert le même service, on peut se passer de cet élément. Le service "PA:RE" de l'élément "*segmentation du message*" peut être fourni par l'élément *"accès au réseau"*. Par contre, le comportement de l'interface n'est alors plus même, la tâche se bloquant en attendant le transfert complet du message. Les éléments ne sont donc pas choisis uniquement en fonction des services à fournir mais en fonction du comportement désiré. Le service sert d'interface aux éléments, et sert à tester leur compatibilité pour leur assemblage. C'est le choix de l'élément qui fixe le comportement. Le problème est similaire aux modèles en couches présentés au chapitre 2, où des couches requérant et fournissant le même service peuvent utiliser différents protocoles. Ce modèle doit donc être écrit par l'utilisateur en fonction des caractéristiques désirées.

3.4 Raffinement du système et découpage logiciel/matériel des communications

3.4.1 Principe du raffinement d'une architecture virtuelle

Le système peut être modélisé à un haut niveau d'abstraction par une architecture virtuelle, mais au niveau transfert de registres, la communication se fait par l'intermédiaire d'une interface logiciel/matériel. Pour profiter pleinement des avantages d'un modèle de haut niveau du système, il serait très intéressant de disposer d'un outil permettant de raffiner un modèle de haut niveau du système en un modèle RTL. La réalisation d'un tel outil implique d'être capable de générer automatiquement les interfaces logiciel/matériel.

Le problème de la conception de ces interfaces commence par leur spécification, qui se fait à partir de la spécification des ports virtuels. Comme nous l'avons vu dans le chapitre 3.2, un protocole peut être modélisé par un graphe de dépendances de services. Les relations de dépendance entre les services peuvent être utilisées pour configurer automatiquement ce protocole en fonction de l'application. La méthode proposée par [Zit93] pourrait être utilisée pour la génération de la spécification des ports virtuels. En revanche, [Zit93] ciblait des plateformes multiprocesseurs comme architecture cible pour son sous-système de communication. Dans un système monopuce, il est complètement inenvisageable d'utiliser un ou plusieurs processeurs pour le contrôle du protocole. On n'utilise pas plusieurs processeurs pour permettre la communication d'un processeur exécutant une partie de l'application. Le

coût serait bien trop important, d'autant que les protocoles utilisés ne sont pas les mêmes et ne requièrent pas la même puissance de calcul. De plus, les systèmes monopuces sont destinés à des systèmes spécifiques, et il n'est donc pas indispensable d'avoir des communications aussi flexibles que dans un réseau.

Comme il l'a déjà été présenté, l'architecture logiciel/matériel cible se compose d'un SE, d'un HAL, d'un sous-système processeur et de l'interface (figure 3.12).

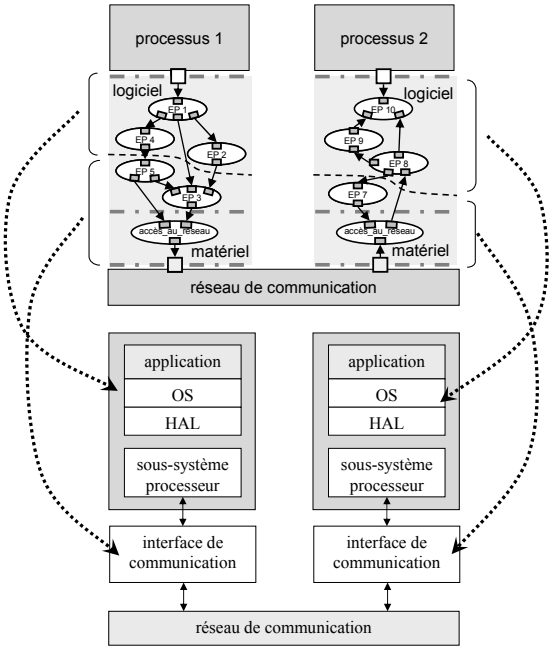


Figure 3.12 - Le flot de raffinement

Les communications sont supportées dans l'interface et dans les couches logicielles. Le flot vise donc à découper le modèle en une partie logicielle et une partie matérielle puis à générer le code correspondant. La partie matérielle correspond à l'interface de communication.

3.4.2 Le flot de raffinement

a) Présentation du flot de raffinement

Le flot de synthèse des interfaces logiciel/matériel en cours d'élaboration au sein du laboratoire TIMA part de l'architecture virtuelle du système. Dans cette architecture, l'enveloppe requiert et fournit des services à l'application et au réseau de communication. Cette enveloppe, comme nous l'avons vu, peut être modélisée par un graphe de dépendance de services. L'idée de base du flot est d'appliquer les méthodes de génération présentées au chapitre 3.2 à la génération de l'enveloppe.

La figure 3.13 présente un flot possible pour générer les interfaces logiciel/matériel à partir de la spécification des ports virtuels. Ce flot n'en est encore qu'au stade de la recherche mais le travail réalisé pendant cette thèse s'inscrit à l'intérieur de celui-ci et vise à répondre à certains problèmes posés par celui-ci. Les travaux menés par cette thèse ont donc été menés en parallèle avec d'autres thèses au sein du groupe SLS. Ce chapitre vise à définir en quoi les travaux menés durant cette thèse

sont complémentaires avec d'autres travaux, en vue de présenter une solution globale au problème de la génération automatique des interfaces logiciel/matériel.

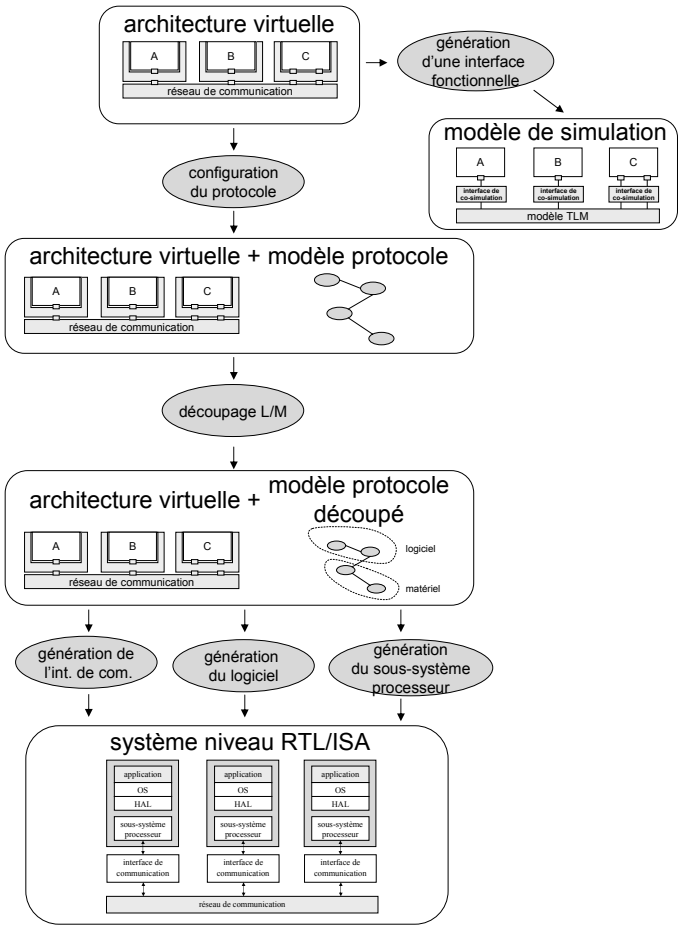


Figure 3.13 - Le flot de génération des interfaces de communication

Le flot part d'une architecture virtuelle du système. Celle-ci peut être simulée grâce à la génération automatique d'un modèle de simulation. La génération de l'interface logiciel/matériel passe alors par la génération automatique de la spécification des ports virtuels. Il est ensuite nécessaire de choisir les fonctions à réaliser en logiciel ou en matériel dans l'étape de découpage logiciel/matériel. A partir de là, sont générés l'interface de communication, le logiciel et le sous-système processeur.

b) Application de la génération d'un SDG à la simulation des interfaces L/M

[Sar05] a appliqué le principe de [Gau01] pour la génération de modèles fonctionnels de l'interface logiciel/matériel destinés à la simulation d'architectures virtuelles. A partir d'un modèle de l'architecture virtuelle où sont spécifiés les services offerts et requis par l'application et le réseau, un outil génère automatiquement un modèle fonctionnel simulable de l'enveloppe.

c) Configuration du protocole

[Sar05] a travaillé uniquement sur des modèles fonctionnels de l'enveloppe. L'ajout de critères aux services permettrait la configuration de protocoles, en prenant en compte des critères non fonctionnels

tel que la performance. L'application de la méthode de [Zit93] à notre cas permettrait la génération de la spécification des ports virtuels, en supposant des protocoles et des critères de services différents.

d) Le découpage logiciel/matériel

La spécification des ports virtuels décrit les services que l'interface logiciel/matériel doit fournir au logiciel pour communiquer avec les autres modules par l'intermédiaire du réseau de communication. A partir du modèle du protocole, notre objectif est d'obtenir un pilote de communication et une interface de communication. Il doit donc être découpé en une partie logicielle et matérielle afin de choisir les fonctionnalités qui sont implémentées par le pilote de communication et celles qui sont réalisées par l'interface de communication. [Pav04] a travaillé sur ce problème du découpage logiciel/matériel.

Les services de communication offerts à l'application étant implémentés à la fois en matériel et en logiciel, il est possible d'offrir un compromis coût/performance au concepteur en agissant sur le découpage logiciel/matériel des communications. La réalisation matérielle si elle offre plus de performance est en revanche généralement moins flexible et plus coûteuse (dans le cas de la communication). Une réalisation logicielle est plus flexible, mais elle est généralement moins performante qu'une solution matérielle et consomme plus. Il faut également tenir compte de l'interaction entre le logiciel et le matériel, pour déterminer les performances globales de la communication. Un trop grand nombre de synchronisation par interruptions consomme par exemple de la ressource processeur.

[Pav04] a recherché une méthode pour choisir quelle réalisation (logiciel ou matériel) doit être adoptée pour chaque élément de protocole. Il a également réfléchi au lien entre ce découpage et la génération des pilotes. La contribution de cette thèse au découpage logiciel/matériel est de faire le lien entre le découpage du graphe et la génération des interfaces. Le découpage logiciel/matériel des communication ne consiste en effet pas uniquement à faire des choix de réalisation mais implique aussi d'introduire le processeur, et de définir comment le logiciel et le matériel vont communiquer ensemble. Pour cela, un modèle de spécification de l'interface de communication a été défini et est présenté plus loin.

e) Génération de sous-systèmes processeurs

[Lyo03] a traité de la génération des sous-systèmes processeurs; le nom du sous-système processeur à utiliser est annoté dans l'architecture virtuelle. Il existe également des outils commerciaux [Men04].

f) Génération des pilotes de communication et du système d'exploitation

Le chapitre 3.2 a déjà présenté le travail de [Gau01] sur la génération de systèmes d'exploitations. [Pav04] a proposé une extension à ce travail pour introduire le modèle de communication présenté et son découpage logiciel/matériel.

g) Génération automatique de l'interface

Le découpage logiciel/matériel permet de choisir quels éléments de protocole sont réalisés en logiciel ou en matériel. L'ensemble des éléments de protocole à réaliser en matériel fait partie de la spécification de l'interface de communication. Un outil présenté dans le chapitre 4 génère un modèle RTL de l'interface à partir de cette spécification. Cette étape du flot est une des contributions de cette thèse.

3.4.3 Spécification de l'interface de communication

L'interface de communication connecte le sous-système processeur au réseau de communication. Elle est donc connectée au sous-système processeur par un ensemble de ports. Cet ensemble de ports physiques est modélisé dans la spécification de l'interface par un unique port abstrait, appelé P_SBSYS, représentant l'ensemble des ports. Le nom de ce port a été figé afin de le distinguer des ports avec le réseau, pour des raisons propres à la méthode de génération des interfaces décrite au chapitre 4. Les ports de l'interface avec le réseau sont eux spécifiés dans l'architecture virtuelle.

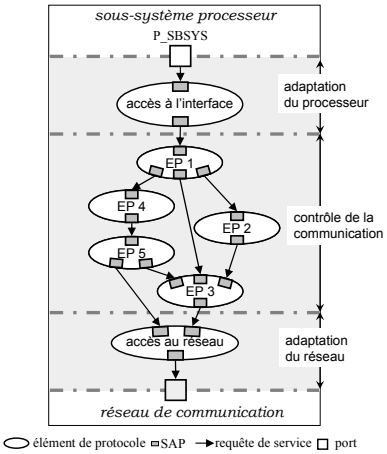


Figure 3.14 - La spécification de l'interface de communication

La spécification de l'interface est composée entre autre du sous-ensemble des éléments de protocole composant la spécification des ports virtuels, et devant être réalisés en matériel. Ce sous-ensemble spécifie le comportement de l'interface. Cette partie a été nommée "contrôle de la communication" sur la figure 3.14 car elle est responsable de la gestion du protocole. On distingue deux autres parties dans la spécification de l'interface. La partie d'adaptation au réseau qui était également présente dans la spécification des ports virtuels représente l'accès physique de l'interface au réseau de communication (contrôle des signaux, adaptation des domaines d'horloges). Un élément de

protocole “accès au réseau” est connecté à chaque port du réseau de communication. Au moment du découpage logiciel/matériel, une partie adaptation du processeur doit être ajoutée. Elle est composée d'un seul élément de protocole “accès à l'interface” représentant l'accès à l'interface du sous-système processeur. Les services offerts par cet élément sont les mêmes que ceux qu'il requiert. Le rôle de cet élément est lié à la méthode de génération et sera donc explicité dans le chapitre suivant.

La spécification de l’interface de communication correspondant à l’exemple précédent (chapitre 3.3.3.c) est représentée sur la figure 3.15

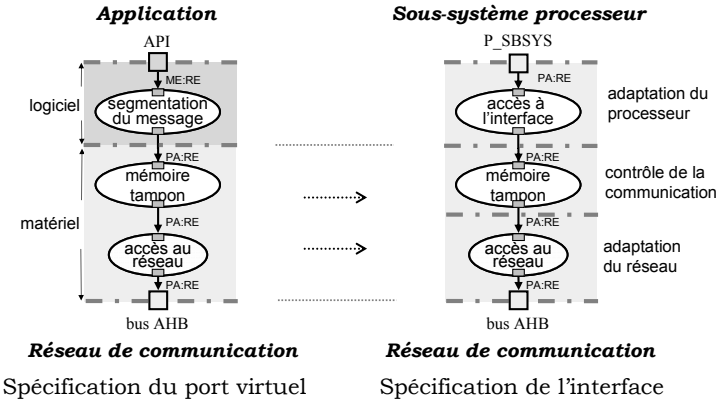


Figure 3.15 - De la spécification des ports virtuels à la spécification de l’interface de communication

Même si une perspective de mon travail est qu'il soit intégré dans le flot présenté, il a néanmoins été pris comme hypothèse que la spécification était écrite à la main par l'utilisateur. Elle peut être utilisée pour des interfaces de communication destinées à connecter aussi bien des sous-systèmes processeurs que des IP. Dans le cas des interfaces destinés à des sous-systèmes processeurs, la génération de cette spécification à partir du découpage logiciel/matériel reste un problème ouvert.

3.5 Conclusion

Ce chapitre a défini un modèle de spécification des interfaces de communication. La notion d'interface logiciel/matériel a été définie ainsi qu'un flot de génération des interfaces logiciel/matériel. En intégrant la génération des interfaces de communications, la méthodologie doit permettre de répondre aux problèmes posés par l'intégration logiciel/matériel et doit permette une réalisation mixte logiciel/matériel de la communication. Un modèle de spécification de l'interface ayant été défini, la problématique de cette thèse consiste à générer une interface de communication à partir de sa spécification. Comme nous avons fait l'hypothèse au chapitre 2 qu'une approche par assemblage de composants réutilisables était plus efficace, des composants doivent être sélectionnés dans une bibliothèque puis assemblés pour générer l'interface à partir de la spécification. Le chapitre suivant détaille la méthodologie utilisée à cette fin.

Chapitre 4

Génération automatique des interfaces de communication

Sommaire

4.1 Introduction

Comme nous venons de le voir, les concepts de services et d'enveloppes facilitent l'assemblage des composants du système à un haut niveau d'abstraction. Mais pour être effective, la méthodologie doit reposer sur des outils de génération de l'interface logiciel/matériel, permettant le raffinement de l'architecture virtuelle du système en un modèle RTL. Ce chapitre présente une méthodologie de génération des interfaces de communication.

4.2 Principe de la méthodologie de génération automatique

Une des difficultés de la conception d'une interface est la définition de son architecture. Une solution simple pour générer automatiquement l'interface est d'utiliser la structure de la spécification pour guider l'assemblage des composants RTL. La génération de l'interface se réduit alors à remplacer chaque élément de protocole par un composant matériel au niveau RTL. Cela offre également l'avantage de pouvoir contrôler l'architecture de l'interface en écrivant la spécification.

Le raffinement de la spécification jusqu'au niveau transfert de registres se fait donc en sélectionnant un composant RTL pour chaque élément de protocole dans une bibliothèque, et en remplaçant l'élément de protocole par le composant RTL choisi. Les éléments de protocole étant configurables, les composants matériels correspondant aux éléments de protocole le sont donc logiquement aussi. Ils sont configurés en fonction de l'application (ex. : taille de bus, taille des FIFO). Ces composants sont alors assemblés pour obtenir l'interface. Le résultat en sortie de notre méthodologie est un modèle RTL synthétisable de l'interface. Les composants sont écrits avec un langage de description de matériel (VHDL, Verilog, SystemC). Des macros sont utilisés pour permettre la configuration du code. Les composants sont rangés dans une librairie.

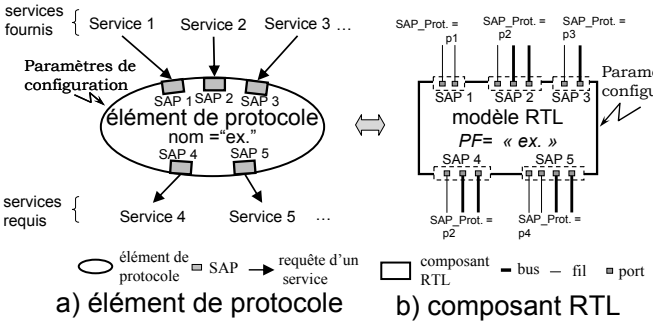


Figure 4.1 - Le modèle des composants

La figure 4.1 représente un élément de protocole et le composant RTL réalisant la fonctionnalité de cet élément de protocole. Au niveau RTL, les Points d'Accès aux Services (SAP) deviennent un ensemble de ports. Un paramètre "SAP protocole" attaché à cet ensemble de ports indique le protocole utilisé par les composants RTL pour communiquer avec les autres composants. Cet ensemble de ports est regroupé à l'intérieur d'un port hiérarchique, que l'on appelle port macro. Le paramètre "SAP protocole" est attaché à ce port. La figure 4.2 illustre le principe du raffinement de la spécification.

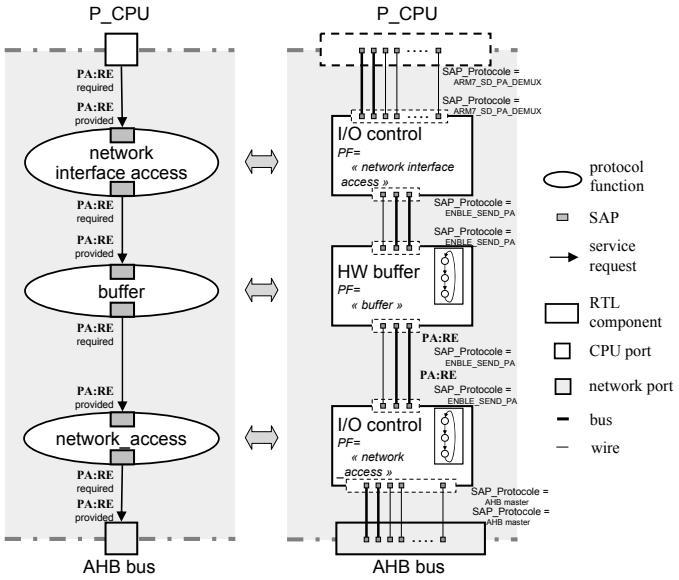


Figure 4.2 - Principe du raffinement

Les composants de bibliothèques peuvent être hiérarchiques, et être eux-mêmes composés de sous-modules. Le choix des sous-modules et leur assemblage restent néanmoins à la charge du concepteur de bibliothèques. Cela implique que le modèle d'un composant doit décrire les sous-modules utilisés, fixer les paramètres de configuration de ces composants, et les interconnecter. Si le choix des sous-modules n'est pas figé, les possibilités de sélection sont malgré tout restreintes par les possibilités du langage de description (impossibilité de faire une recherche de composants dans la bibliothèque).

4.3 Modèles de représentation

4.3.1 Spécification de l'interface de communication

La spécification comprend une description des ports de l'interface avec le réseau, et le sous-système processeur. Le port de l'interface avec le sous-système processeur doit s'appeler "P_SBSYS". Ceci permet à l'outil de génération de le distinguer des autres ports. La génération des interfaces devant être intégrée dans un flot de génération des interfaces logiciel/matériel, fixer le nom permet de faciliter l'automatisation de la connexion de l'interface avec le sous système processeur. Les noms des ports de l'interface avec le réseau sont les noms utilisés dans l'architecture virtuelle du système.

La spécification doit également décrire les éléments de protocole utilisés, les paramètres de configuration et le lien entre les SAP des éléments de protocole. La définition des services offerts ou requis par chaque SAP est utile pour l'écriture de la spécification mais n'intervient pas dans la génération de l'interface.

4.3.2 Eléments de bibliothèque

Le modèle d'un composant de bibliothèque est découpé en deux parties : une partie comportementale et une partie déclarative. La partie comportementale correspond au modèle HDL du

composant. La partie déclarative correspond à la déclaration du composant dans la bibliothèque, qui sert aussi à la déclaration du composant dans le modèle RTL de l'interface (c'est-à-dire le modèle d'assemblage des composants, son schéma électrique, on parle de "netlist RTL" en anglais).

Le modèle de bibliothèque décrit le nom des composants, ses ports, le nom de l'élément de protocole réalisé ("implémenté"), les relations entre les paramètres de l'élément de protocole et les paramètres du composant. La correspondance entre les ports SAP et les ports RTL est assurée par la concordance des noms des ports macro et des ports SAP.

Le code HDL des composants est synthétisable avec les outils de synthèse logique classiques (ex. : [Syn01b]). Il est écrit dans un langage HDL avec des macro pour obtenir des composants configurables. Les paramètres de configuration sont propres à chaque composant.

4.4 Flot de raffinement

4.4.1 Présentation du flot

Notre méthodologie raffine la spécification de l'interface de communication en un modèle RTL, pouvant être utilisé pour la synthèse logique à partir des outils existants. Nous avons développé un outil, décrit dans le chapitre 4.5, basé sur cette méthodologie.

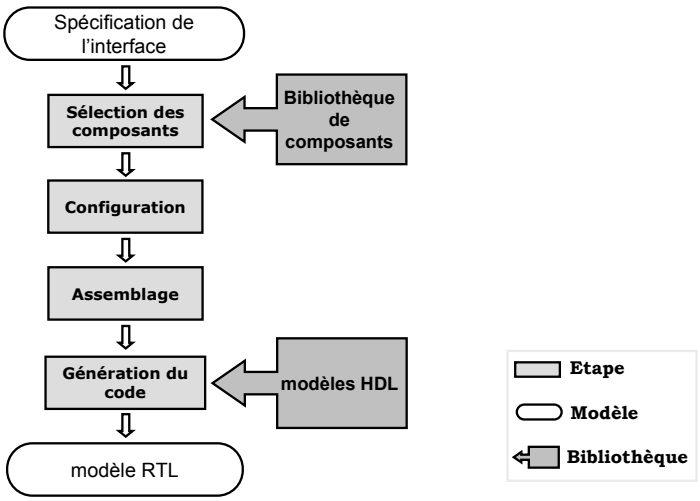


Figure 4.3 - Le flot de conception

Le flot de conception est composé de quatre étapes (figure 4.3) :

a) <u>la sélection des composants</u> : un composant RTL est sélectionné pour chaque élément de protocole. La sélection doit être faite de façon à avoir des composants compatibles.

b) <u>la configuration</u> : Les composants RTL sont configurables. L'outil extrait de la spécification des paramètres qui sont utilisés pour déterminer les paramètres de configuration de chaque composant.

c) <u>l'assemblage des composants</u> : les éléments de protocole sont remplacés par les composants RTL sélectionnés et paramétrés. L'assemblage consiste à remplacer les éléments de protocole par les composants RTL sélectionnés et à générer les interconnexions.

d) <u>la génération du code</u> : un modèle RTL de l'interface écrit dans un langage HDL courant (VHDL, Verilog, SystemC) est ensuite généré et peut être utilisé avec les outils de synthèse logiques existants.

4.4.2 La sélection des composants

La première étape du flot de conception est la sélection des composants. Nous faisons l'hypothèse que chaque élément de protocole dispose d'au moins une réalisation matérielle dans une bibliothèque de composants RTL. Tout composant RTL réalisant la fonctionnalité d'un élément de protocole est approprié (pour le remplacer). Il faut donc sélectionner des composants RTL réalisant chaque élément de protocole de la spécification de l'interface.

La sélection des composants doit tenir compte de la compatibilité des composants entre eux. Une liste des composants RTL pouvant remplacer chaque composant est d'abord obtenue. L'outil recherche alors l'ensemble des combinaisons de composants RTL compatibles pour générer l'interface. C'est alors au concepteur de choisir une solution.

a) Choix des composants

Tout d'abord, la sélection des composants se fait de manière à respecter le comportement attendu de l'interface, tel que décrit dans la spécification. Un composant RTL réalise en matériel la fonctionnalité d'un élément de protocole, c'est une réalisation matérielle de cet élément de protocole. La définition du composant RTL (dans la bibliothèque) inclut le nom de l'élément de protocole implémenté. Ce nom est attaché sous la forme d'un paramètre à chaque composant RTL pour indiquer quel élément de protocole est réalisé. Tout composant RTL ayant le bon paramètre peut être utilisé. Cela permet de sélectionner un sous-ensemble de composants RTL qui peut réaliser le comportement désiré. Si il existe plusieurs alternatives de composants RTL pour un élément de protocole, il faut donc sélectionner un unique composant RTL pour obtenir la meilleur combinaison de composants réalisant l'interface.

Dans l'exemple suivant, on a un port d'entrée P1 et un port de sortie P2 avec des canaux de communication point à point. Le protocole utilisé pour les canaux de communication est un protocole rendez-vous ("handshake") à quatre phases. La synchronisation des communications est assurée par l'intermédiaire d'une FIFO implémentée dans l'interface de communication du module émetteur. La figure 4.4.a représente la spécification de l'interface de communication de notre exemple, et la figure 4.4.b montre un exemple de composants pouvant se trouver dans la bibliothèque et la présélection réalisée à cette étape.

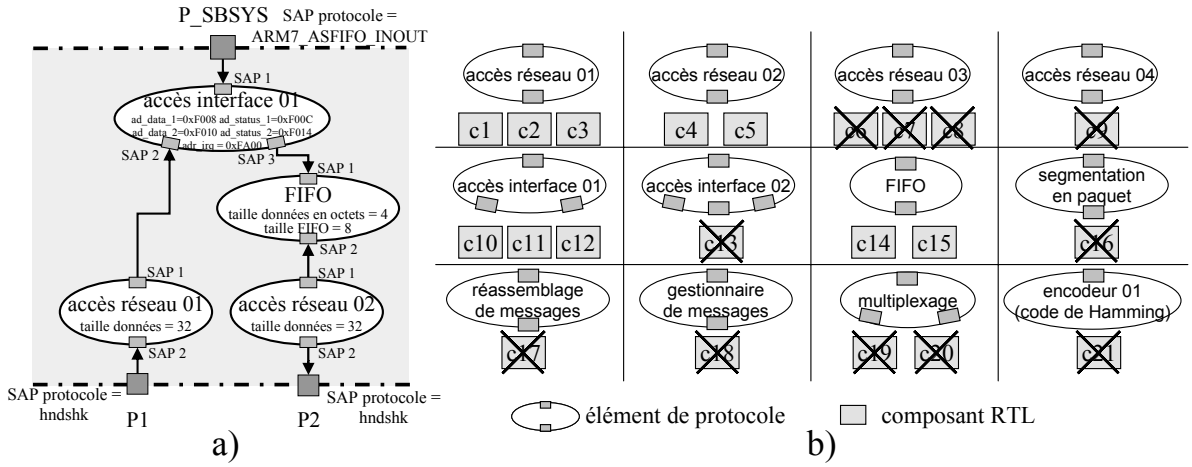


Figure 4.4 - Choix des composants implémentant les éléments de protocole

Sur la figure 4.4.b), les composants RTL rangés dans la même case qu'un élément de protocole sont des réalisations matérielles de cet élément. Puisque l'élément de protocole “accès réseau 01” est utilisé dans la spécification de l'interface, les composants C1, C2, C3 sont susceptibles d'être utilisés pour générer l'interface. L'élément de protocole “segmentation en paquet” n'est par contre pas utilisé. Le composant C16 n'a donc aucun intérêt pour notre exemple. Il est donc barré sur la figure pour indiquer qu'il n'est pas sélectionné. Les composants non barrés constituent le sous-ensemble de composants sélectionnés après le premier processus de sélection.

b) Choix des protocoles

A cette fin, on vérifie ensuite que ces composants RTL sont compatibles. Ils doivent être compatibles en terme d'interface physique et en terme de protocole (format des données échangées, synchronisation). Ceci est décrit par le paramètre “SAP protocole”. Quand un service est fourni par un élément de protocole à un autre, les ports correspondant des composants RTL doivent avoir le même “SAP protocole”.

Cette sélection se fait d'abord en fonction du SAP protocole des ports de l'interface comme on peut le voir sur la figure 4.5. L'élément de protocole “accès interface 01” dispose de trois réalisations matériels C10, C11, C12. Le SAP “SAP 1” de cet élément est connecté au port P_SBSYS. Le paramètre SAP_protocole attaché à ce port sert de critère de sélection. Pour chacun des composants C10, C11, C12, on regarde si le paramètre SAP_protocole attaché au port macro correspondant à SAP 1 est le même que celui de P_SBSYS. On a donc sélectionné C10 et exclu C11 et C12 du processus de sélection.

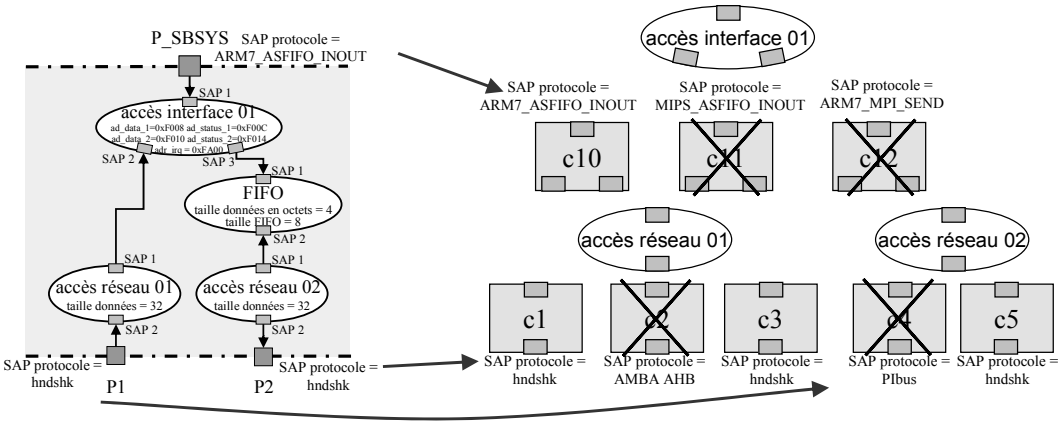


Figure 4.5 - Choix des protocoles (1)

La sélection se fait ensuite en fonction de la compatibilité des éléments entre eux (figure 4.6).

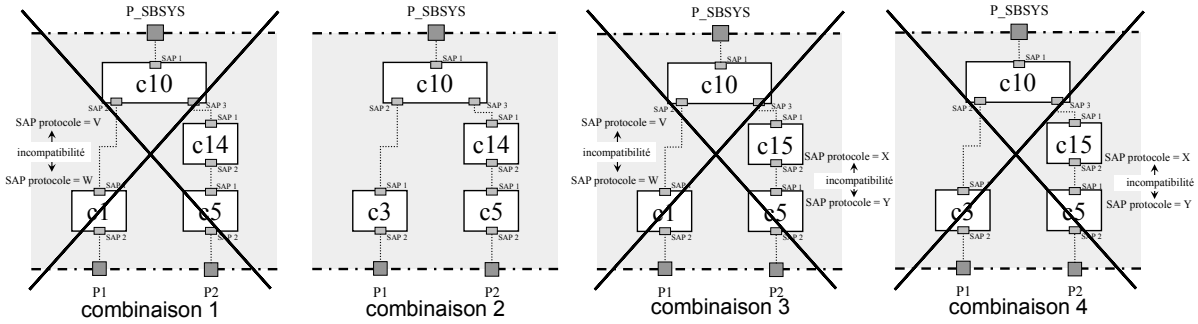


Figure 4.6 - Choix des protocoles (2)

Dans notre cas, on a deux éléments de protocole (“accès réseau 01” et “FIFO”) qui ont encore chacun deux choix de réalisation matérielle. On peut donc réaliser quatre combinaisons de choix de composants possibles. Dans la spécification de l’interface, le SAP “SAP 2” de l’élément “accès interface 01” est connecté au SAP “SAP 1” de l’élément de protocole “accès réseau 01”. Le port macro SAP 2 du composant C10 devrait donc être connecté au port macro SAP 1 du composant C1. Comme les paramètres “SAP_protocole” attachés à ces deux ports sont différents, cette connexion est impossible et la combinaison n’est pas valide. Ces deux paramètres définissent en effet les signaux et le protocole associé utilisés par chaque composant pour fournir ou utiliser les services des SAP.

c) Choix des domaines d'horloges

Le troisième critère à vérifier lors de la sélection d'un composant RTL est qu'ils utilisent la même horloge. En effet, les systèmes monopuces peuvent être Globalement Asynchrone, Localement Synchrone. Il se peut donc que le réseau de communication et l’unité de traitement à connecter utilisent des horloges différentes, et par conséquence certains composants de l’interface utiliseront l’horloge du réseau et d’autres l’horloge de l’unité de traitement. Puisque deux composants synchrones communiquant avec un protocole synchrone ne peuvent pas fonctionner avec des horloges différentes, il est nécessaire de prendre en compte les domaines d’horloge lors de la sélection des composants.

Les domaines d'horloge de chaque composant sont déterminés à partir des règles suivantes. Deux composants peuvent être connectés s'ils appartiennent au même domaine d'horloge. S'ils utilisent deux horloges différentes, ils peuvent être connectés si la communication entre les deux est asynchrone (indiqué par un paramètre attaché avec le "SAP protocole"), ou si au moins l'un des deux composants supportent deux horloges (le nombre d'horloges qu'un composant peut utiliser est restreint à deux, une pour le réseau et une pour le sous-système processeur).

L'adaptation des domaines d'horloge n'est donc pas gérée par un composant particulier au niveau de l'interface. Malgré tout, l'élément de protocole "accès au réseau" représente l'adaptation physique entre le réseau et le sous-système processeur. Il serait donc logique que l'adaptation des domaines d'horloge soit gérée au niveau du composant réalisant cet élément de protocole, dans un souci de réutilisation des composants.

Les domaines d'horloges choisis pour les composants de l'exemple précédent sont annotés sur la figure 4.7.

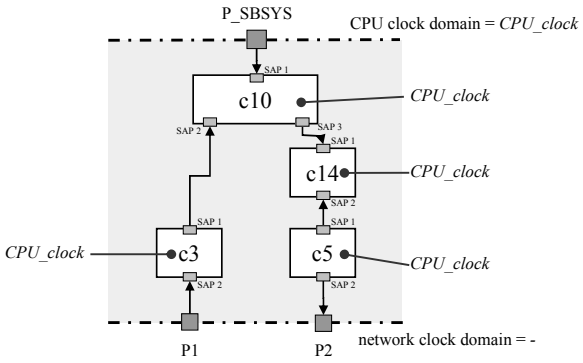


Figure 4.7 - Choix des domaines d'horloge

Le domaine d'horloge du sous-système processeur est défini par le paramètre "CPU clock domain" et s'appelle dans notre exemple "CPU_clock". Les canaux de communication utilisant un protocole asynchrone, aucun domaine d'horloge n'a été défini pour le réseau. Le paramètre "network clock domain" n'a donc pas de valeur assignée. Dans le cas de notre exemple, la sélection des domaines d'horloge est triviale car un seul domaine d'horloge est défini. Chaque composant utilise donc le domaine d'horloge du processeur, et on a donc annoté la chaîne de caractères "CPU_clock" pour chaque composant. On obtient ainsi une combinaison unique de composants RTL pour réaliser l'interface, qui sont tous synchronisés avec la même horloge.

d) Selection finale

S'il existe plusieurs combinaisons possibles de composants RTL pour générer l'interface, la méthodologie fait l'hypothèse que c'est au concepteur de choisir la solution la mieux adaptée. A moins d'avoir une bonne connaissance des composants de bibliothèque, le concepteur doit donc évaluer les différentes combinaisons après leur génération en simulant et en synthétisant l'interface. L'obtention d'une solution optimale requiert une démarche systématique pour évaluer les performances de

l'interface (en fonction de la taille et de la fréquence de fonctionnement maximale de l'interface générée, de sa latence, du débit maximal autorisé). La réalisation d'une méthode d'estimation de performances des différentes combinaisons est une perspective intéressante mais qui n'a pas été traitée dans cette thèse.

4.4.3 La configuration des composants

L'étape de configuration du composant correspond à la configuration du modèle de bibliothèque du composant, qui sert à déclarer le composant. Cette étape définit donc les ports du composant ainsi que ses paramètres de configuration. Si le composant est hiérarchique, cette étape s'occupe également de générer l'architecture interne du composant et de configurer ses sous-composants.

Pour cela, cette étape prend en entrée les paramètres de configuration de l'élément de protocole et le modèle de bibliothèque du composant RTL sélectionné pour réaliser cet élément de protocole. Les paramètres de l'élément sont passés en entrée du modèle de bibliothèque du composant qui génère un modèle du composant paramétré et configuré. La figure 4.8 illustre la configuration du composant implémentant l'élément de protocole "FIFO" de l'exemple.

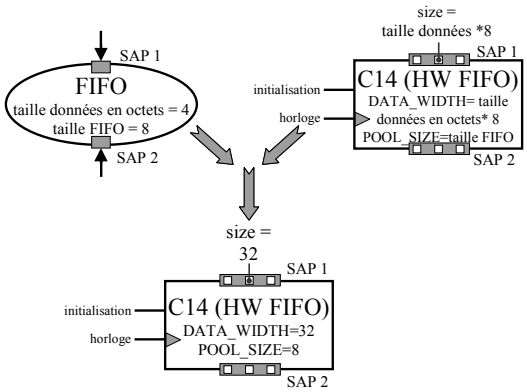


Figure 4.8 - Configuration du composant "HW FIFO"

4.4.4 L'assemblage des composants

L'étape d'assemblage génère la liste des interconnexions ("netlist"). L'architecture de l'interface est donc alors définie au niveau RTL.

Pour cela, chaque élément de protocole est remplacé dans la spécification par le composant correspondant sélectionné puis paramétré. On génère les interconnexions en raffinant les liens entre les SAP des éléments de protocole. Pour chaque SAP de l'élément de protocole, le composant RTL correspondant possède un port hiérarchique du même nom. L'ensemble des sous ports englobés dans un port hiérarchique est décrit par le paramètre SAP protocole. Les relations de dépendance entre services sont remplacées par un ensemble d'interconnexions. Comme on se restreint à des interconnexions point-à-point, cela consiste en la génération des interconnexions entre les ports d'entrée et de sortie correspondant (qui doivent avoir le même nom). Un SAP appelant un service et le SAP fournissant ce service sont remplacés par les mêmes ports. Ceci est assuré au moment de la

sélection des composants RTL grâce au paramètre SAP protocole attaché aux ports hiérarchiques. Chaque port d'un composant a son équivalent sur l'autre composant, excepté que l'un est un port d'entrée et l'autre de sortie. On peut donc générer les interconnexions en fonction du nom des ports.

Il est également nécessaire de connecter les signaux d'horloges et d'initialisation (reset) indispensables au fonctionnement des composants. Chaque composant possède des ports à connecter aux signaux d'horloge et d'initialisation. Au moment, du raffinement on rajoute des ports de ce type à l'interface et on crée des interconnexions entre ces ports et tous les composants de l'interface.

Il est également nécessaire de raffiner les ports de l'interface. Le raffinement des ports se fait par copie des ports du composant RTL connecté au port de l'interface.

La figure 4.9 représente l'assemblage des composants de l'exemple précédent.

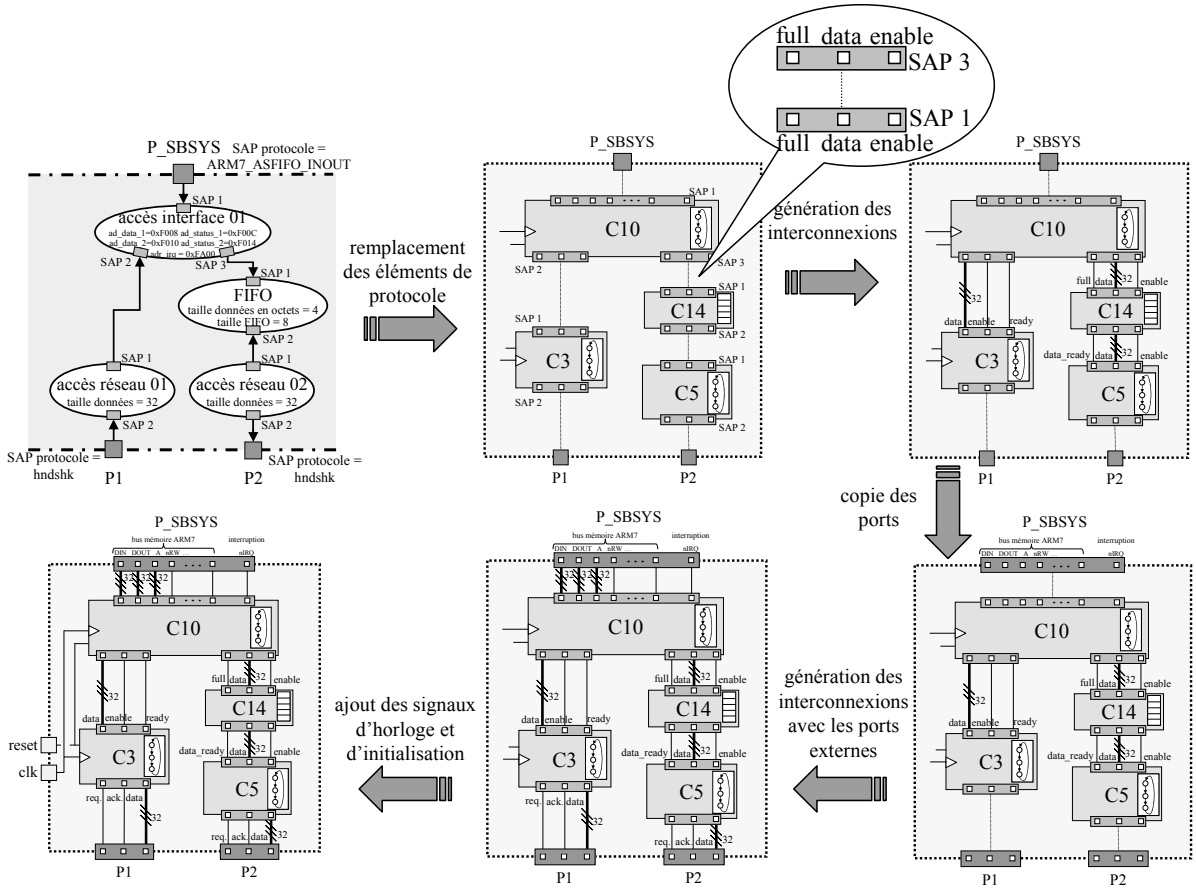


Figure 4.9 - L'assemblage des composants

4.4.5 Génération de code

La dernière étape est la génération du code HDL des composants. En sortie du flot, nous obtenons alors un modèle RTL de l'interface décrit avec un HDL. Comme nous l'avons vu précédemment, chaque composant possède un modèle écrit avec un langage macro dans une bibliothèque. Il est utilisé pour générer le code HDL de chaque composant, qui s'obtient par expansion des macro du modèle. Il est nécessaire d'utiliser un outil d'expansion propre au langage macro utilisé. Il prend en entrée les paramètres de configuration calculés pendant l'étape de configuration.

La figure 4.10 montre l'exemple de la génération d'une partie du code du composant implémentant la FIFO.

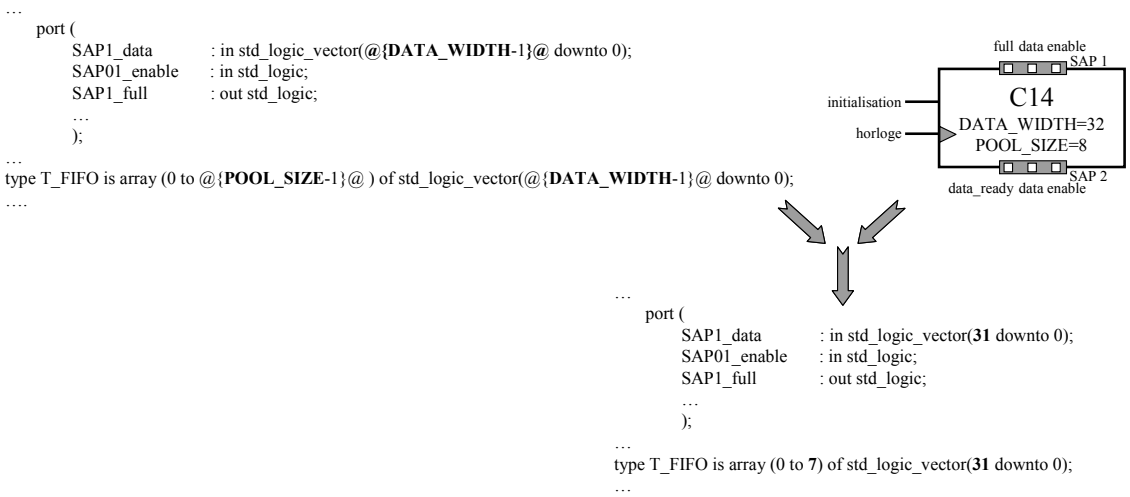


Figure 4.10 - La génération de code

4.5 Développement d'un outil de génération automatique des interfaces de communication

4.5.1 Langages de modélisation

a) Préliminaire : description de Colif

Pour réaliser un outil de génération automatique basé sur la méthodologie présentée, il était nécessaire d'avoir un format informatique pour modéliser aussi bien des composants matériels que des éléments de protocole. Colif ("CO-design Language Independent Format") est un modèle orienté objet de représentation de systèmes hétérogènes développé au sein de l'équipe SLS [Ces01]. Colif permet une modélisation hétérogène en langages et/ou niveaux d'abstraction. Les trois concepts de base de Colif sont les modules, les ports et les "nets" (figure 4.11). Colif représente un système comme un ensemble de modules hiérarchiques interconnectés par un réseau de communication composé de nets et de ports hiérarchiques.

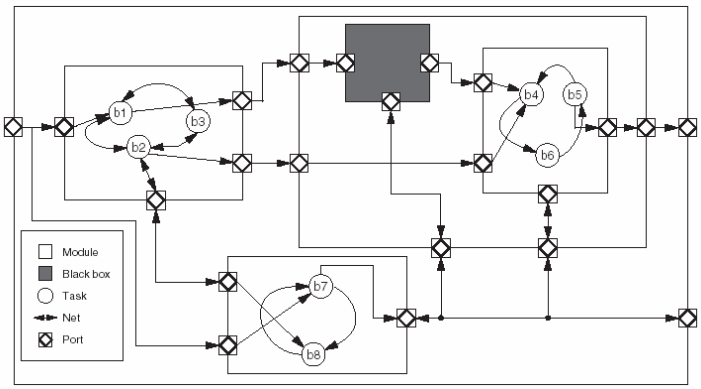


Figure 4.11 - Représentation graphique de Colif

Chaque module est défini par son contenu et son interface, qui consiste en un ensemble de ports. Le contenu d'un module définit sa structure interne ou contient une liste de descriptions comportementales. Un port modélise la connexion entre l'intérieur d'un module et son environnement externe. Un port peut être hiérarchique. Un “net” représente un média de communication entre un ou plusieurs ports. Les détails de réalisation tels que les signaux spécifiques au protocole de communication peuvent être cachés par une définition hiérarchique des “nets” offrant ainsi une abstraction des canaux de communication. Les modules, ports et “nets” peuvent être définis à différents niveaux d'abstraction. Ainsi, un “net” peut représenter aussi bien un canal de communication virtuel que des interconnexions physiques.

Chaque concept est décliné en plusieurs classes. La figure 4.12 est une représentation simplifiée du diagramme UML [UML] des classes de Colif.

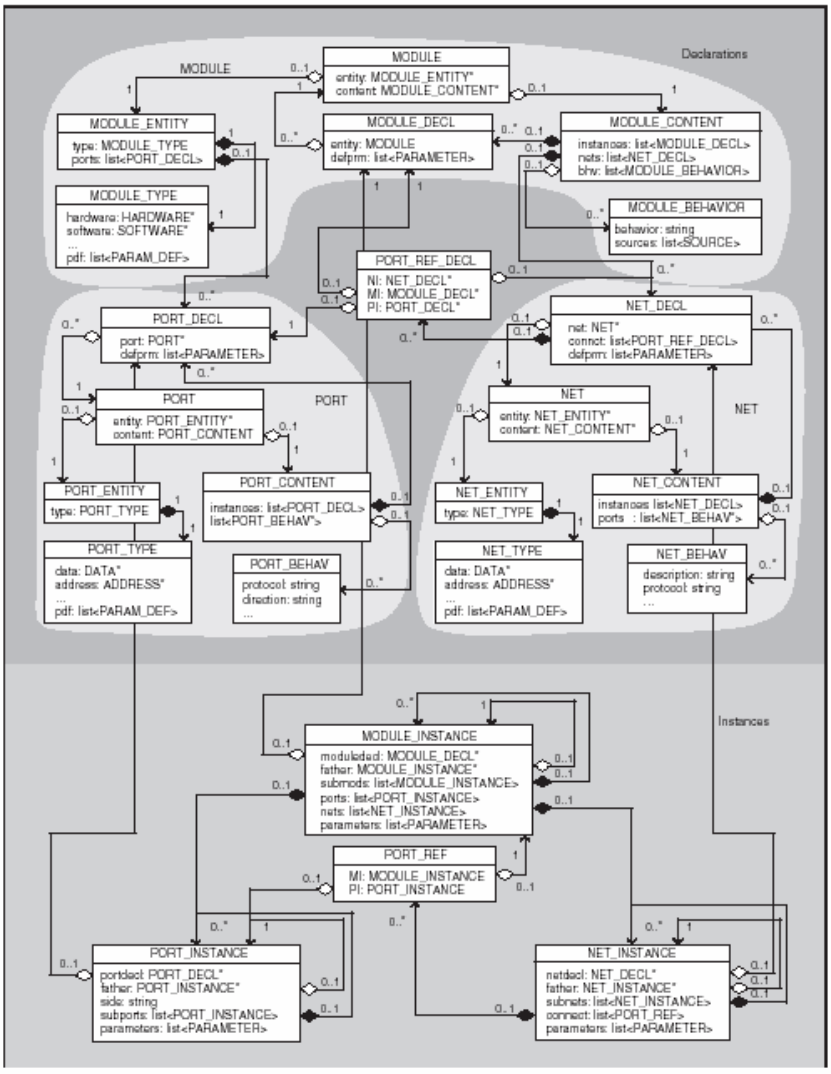


Figure 4.12 - Diagramme UML des classes de Colif

Les classes MODULE, NET et PORT servent à la définition de modules, nets et ports. Les classes MODULE_DECL, PORT_DECL et NET_DECL sont utilisées pour déclarer l'utilisation de sous-

modules, sous-ports et sous-nets. Les définitions sont utilisées par les classes MODULE_INSTANCE, PORT_INSTANCE et NET_INSTANCE.

Une construction en couches a été utilisée pour la réalisation de Colif. Au plus haut niveau, un ensemble de classes C++ est utilisé pour créer et manipuler des objets Colif. Les modèles Colif sont sauvegardés dans des fichiers au format XML [XML].

b) Spécification en Colif générée à partir de Vadel

L'architecture virtuelle du système est écrite en Vadel (“Virtual Architecture Description Language”). Vadel est une extension de SystemC [OSCI] pour la description des architectures virtuelles développée au sein de l'équipe SLS du laboratoire TIMA. Après compilation, un programme Vadel génère à l'exécution un modèle en Colif du système. [Nic02, Sar04] ont travaillé sur un outil de simulation de ce système prenant en entrée le modèle Colif généré.

c) Spécification des interfaces en Colif

La spécification de l'interface de communication est également écrite en Vadel afin d'être représentée en Colif. Les éléments de protocole sont représentés par des modules Colif, les SAP par des ports et les dépendances entre les SAP par des nets Colif. Le langage Vadel dans sa version actuelle ne permet pas de définir certains paramètres de la spécification, et il est pour l'instant nécessaire d'ajouter dans le modèle Colif généré ces paramètres manquants. Une évolution de Vadel devrait permettre de corriger ce problème (ajout d'une méthode au langage permettant à l'utilisateur de définir de nouveaux paramètres).

d) Modèles de bibliothèque en Colif générées à partir de python

Au début du chapitre, il a été expliqué que les composants RTL sont rangés dans une bibliothèque structurée en deux parties. La partie déclarative de la bibliothèque est représentée en Colif. Le modèle de déclaration d'un composant sert à la fois pour la création de la bibliothèque, mais aussi pendant l'étape de configuration. Les composants de la bibliothèque sont écrits avec le langage Python. Python [Python] est un langage interprété, orienté objet. Les variables ne sont pas typées. Python est facilement extensible et facilement intégrable à l’intérieur d’un autre programme C++ [Van02]. Python semble donc un choix intéressant pour notre application. La modélisation de composants en Python avait été évoquée comme une perspective dans les travaux de [Lyo03].

Python a été étendu pour manipuler et créer des modèles Colif. L’utilisateur peut très facilement développer ses propres modèles de composant. Pour représenter notre architecture en Python, nous utilisons les concepts habituels que sont les modules, les ports et les nets. De plus, les API cachent certains concepts Colif (tels que les instances) qui ne sont pas nécessaires dans le cas particulier de la création de bibliothèques. Le temps de conception des modèles de bibliothèques est donc plus faible qu'en utilisant les API C++. La figure 4.13 montre un exemple de composant décrit en Python.

```
from Colif import *
from emb import *
from PORTS import *

def IOhndshki_gen(size=32):
    M=Module()

    M.ModEntity("IO_HNDSHK_in","hardware","CH_HW_PF")

    M.Port("nRESET",CPreseti())
    M.Port("CLK",CPclki("bool",1,"node"))
    M.Port("sap1",hndshk_i(size))
    M.Port("sap2",enble_data_i(size))

    M.Source('VHDL','RIV',"$BIBLIO/Lib0.0/PF_impl/Netwrk_Access/HNDSHKi/io_hndshki.vhd.riv")
    M.Source('Colif','Python','$BIBLIO/Lib0.0/PF_impl/Netwrk_Access/HNDSHKi/io_hndshki.py')

    M.Parameters("implemented_PF",'Network_Access02')
    M.Parameters("clk_dmn",'single')
    M.Parameters("DATA_SIZE",size)

    M.ModArch()

    return M


if __name__ == "__main__":
    if emb.option()=='defaut':
        Main=IOhndshki_gen()
        Main.end()
    else:
        Main=IOhndshki_gen(getParameter("DATA_SIZE"))
        Main.end2()
```

paramètre de configuration de l'élément de protocole avec une valeur par défaut

déclaration des ports d'horloge et d'initialisation

déclaration des ports hiérarchiques

fichier du modèle comportemental

référencement de ce fichier

attributs du composant

paramètre de configuration du composant

génération du modèle de bibliothèque avec valeurs par défaut

génération du modèle paramétré

Figure 4.13 - Exemple d'un module décrit en python

Dans l'exemple ci-dessus, les ports SAP1 et SAP2 sont des ports hiérarchiques, eux-mêmes composés d'autres ports. Par exemple, le port SAP1 est un port hiérarchique de type hndshk_i. Un port hndshk_i est défini ainsi :

```
def hndshk_i(size=1):
    P=Port("HNDSHK I","IN","macro","RTL")

    P.ssport("enable",std_logic_in());
    P.ssport("data",std_logic_vector_in(size));
    P.ssport("acknowledge",std_logic_out());

    P.behav("HNDSHKi","P2P","S")

    P.parameter("SAP_protocol","HNDSHKi")
    P.parameter("protocol_asynch","yes")

    return P.end()
```

paramètre de configuration

définition des sous-ports

définition du SAP protocole

Figure 4.14 - Exemple de port hiérarchique défini en Python

Un port hiérarchique est défini par ses sous ports et un paramètre SAP protocole. Dans l'exemple ci-dessus, un paramètre de configuration "size" sert à configurer la taille du sous port "data".

e) Expansion du code HDL

Le modèle de déclaration du composant référence le fichier correspondant au modèle comportemental du composant. Ce fichier a pour but de générer le code source décrivant les différents modules à partir de leur code macro. Les paramètres passés pour l'expansion du code sont les paramètres de la classe ModuleDecl. Tous les paramètres sont passés sans exception. Le langage macro utilisé est le langage Rive [Gau01]. La figure 4.15 (a) montre un exemple de macro-modèle Rive/VHDL. Dans ce code, on souhaite configurer le nom du composant et la taille de deux ports. La figure 4.15 (b) représente le fichier contenant les paramètres de configuration. L'outil d'expansion Rive génère le code final (figure 4.15 (c)).

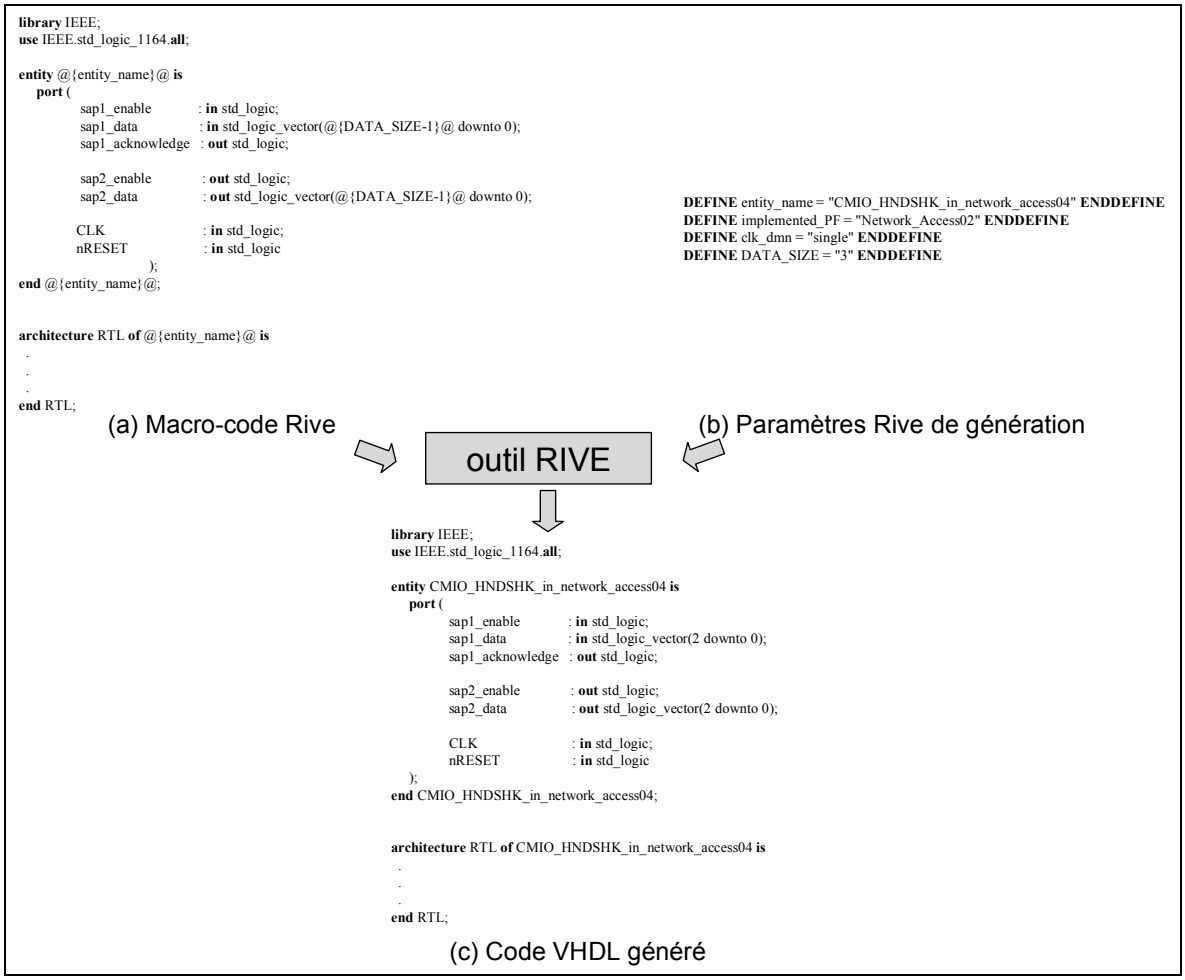


Figure 4.15 - Macro-modèle Rive/VHDL -> VHDL

4.5.2 L'outil ASAG

L'outil ASAG ("Application Specific Architecture Generator") est un outil de raffinement d'architectures virtuelles. Il s'occupe pour cela de la génération de sous-systèmes processeurs, du

raffinement des canaux de communication et de la génération des interfaces de communication [Lyo03]. Cet outil a été modifié dans le cadre de cette thèse pour intégrer la méthodologie de génération des interfaces qui vient d'être décrite. La figure 4.16 représente l'architecture de l'outil avant modification.

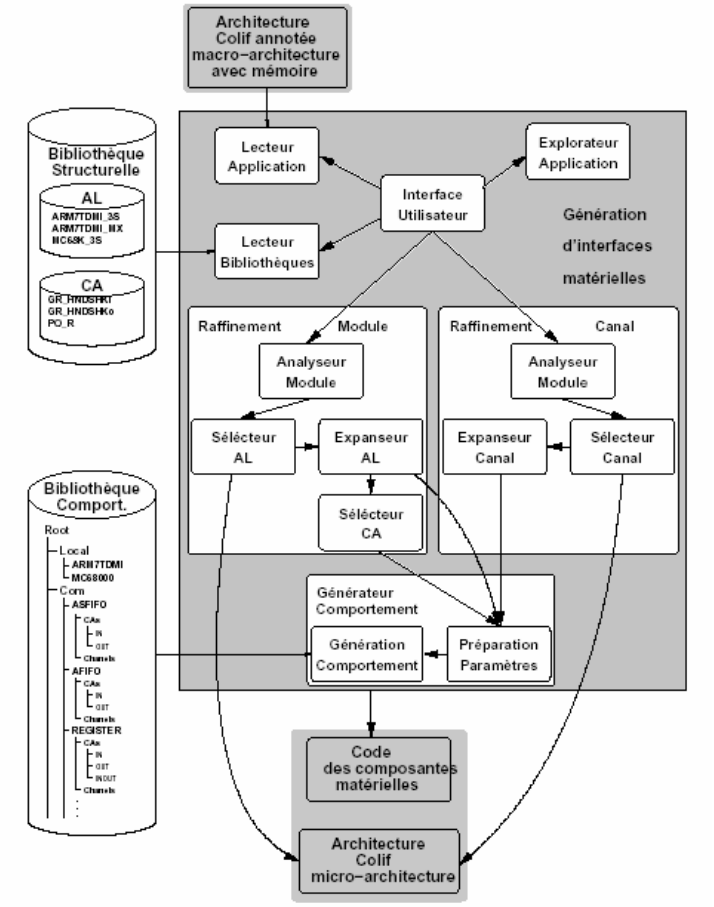


Figure 4.16 - Architecture de l'outil ASAG

Sept modules composent l'architecture de l'outil. Un module "lecteur application" s'occupe de la lecture du modèle Colif de l'architecture virtuelle. Un module "lecteur bibliothèque" est dédié au chargement des bibliothèques. Le modèle d'architecture virtuelle étant hiérarchique, "l'explorateur d'application" permet de parcourir la structure du système pour sélectionner les modules à raffiner. Les modules et les canaux sont raffinés par deux modules distincts ("Raffinement module" et "Raffinement canal"). Un module indépendant est en charge de la génération du code ("générateur comportement"). Enfin, un dernier module sert d'interface avec l'utilisateur ("interface utilisateur").

Dans cet outil, la génération du sous-système processeur et de l'interface de communication se faisaient dans la même étape, correspondant au raffinement du module (figure 4.17 (a)). La méthode utilisée reposait sur le dimensionnement et la configuration d'architectures de bibliothèques. L'interface était pour l'instant modélisée dans l'architecture du sous-système processeur. Jusqu'à maintenant deux paramètres *hardporttype* et *softporttype* attachés aux ports virtuels fixaient la fonctionnalité de l'interface.

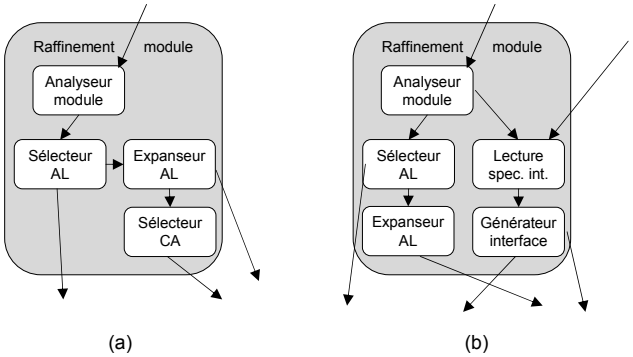


Figure 4.17 - Evolution de l'outil

Dans la nouvelle méthode, on utilise, en plus du modèle de l'architecture virtuelle, la spécification des interfaces de communication comme entrée de l'outil. La figure 4.17 (b) montre l'évolution apportée à l'architecture de l'outil au niveau de l'étape de raffinement du module. Au moment du raffinement d'un module virtuel, l'outil lit la spécification de l'interface puis la génère.

4.6 Analyse : limitations et évolutions futures

4.6.1 Exploration des solutions

En modifiant la spécification de l'interface de communication, on peut gérer différentes architectures pour l'interface et explorer différentes solutions de réalisation. On peut facilement ajuster la spécification de l'interface de communication aux besoins de l'application. Si les composants nécessaires sont disponibles dans la bibliothèque, la nouvelle interface de communication peut être rapidement générée selon la nouvelle spécification.

4.6.2 L'utilisation de bibliothèques

La limitation majeure de la méthodologie est l'utilisation de bibliothèques. La capacité à raffiner une spécification dépend de la disponibilité des bons composants dans la bibliothèque. Si un composant n'est pas disponible dans la bibliothèque, l'utilisateur doit concevoir le composant manquant. L'efficacité de la méthodologie repose sur l'hypothèse que la spécification permette d'optimiser le compromis coût/performance tout en se contentant d'un nombre limité de composants de bibliothèque. L'utilisateur doit pouvoir optimiser l'interface à une application particulière en choisissant et en assemblant les éléments de protocoles parmi ceux disponibles.

Comme concevoir des composants pour une bibliothèque est légèrement plus long que concevoir des composants RTL pour une utilisation spécifique, la capacité à générer une large variété d'interfaces de communication avec seulement un petit ensemble de composants de base est un problème critique.

Le temps d'introduction d'un nouveau composant doit également être court pour permettre à la bibliothèque d'évoluer avec les systèmes qui sont conçus (nouveaux bus, nouveaux réseaux, nouveaux processeurs …).

4.6.3 Décomposition en trois parties

Comme nous l'avons vu dans le chapitre 2, la spécification de l'interface est composée de trois parties. Décomposer l'interface en trois parties (adaptation processeur, protocole, adaptation réseau) favorise la réutilisation des composants.

L'élément de protocole “accès à l'interface” n'a pas de rôle fonctionnel. Il permet une meilleure réutilisation des composants en ayant des éléments de protocole indépendants du sous-système processeur. Décomposer la spécification permet d'avoir une meilleure réutilisation des composants de bibliothèque en ayant des composants indépendants des processeurs ou du choix du réseau ou du protocole. Modifier le réseau requiert uniquement de changer les composants RTL implémentant les éléments de protocole “accès au réseau”.

De plus, la méthode se fonde sur l'hypothèse que des composants de plus faible granularité sont plus réutilisables car ils sont moins spécifiques. La méthodologie ne contraint pas la taille des composants qui sont utilisés.

4.6.4 L'écriture de la spécification

L'assemblage des composants étant automatisée, écrire la spécification des ports virtuels et de l'interface ne requiert pas de connaître les composants disponibles dans la bibliothèque. De plus, pour écrire un composant RTL, le concepteur de bibliothèque a juste besoin de connaître l'élément de protocole qui est réalisé en matériel par le composant. Donc, la spécification d'un protocole de communication et la conception de composants matériels sont deux processus distincts. Cela facilite le support de protocoles de communication compliqués en séparant les considérations liées au protocole et les questions matérielles.

Malgré tout, notre modèle de spécification n'est pas formel. La modélisation implique pour le concepteur de connaître les éléments de protocole disponibles dans les bibliothèques. Un ensemble d'éléments de protocole doit être défini pour que le modèle puisse être facilement réutilisable. La contribution de la méthode se situe dans l’automatisation de l’assemblage des composants RTL. Il est en effet plus facile et plus rapide d’assembler des éléments de protocole que des composants RTL.

De plus, l'architecture de l'interface correspond à la structure de la spécification. La décomposition de la spécification de l'interface de communication en éléments de protocole est faite pour décomposer les fonctionnalités. Mais la décomposition de l'interface de communication en blocs doit être faite selon des considérations matérielles.

Implémenter plusieurs fonctionnalités dans un unique composant RTL peut être nécessaire pour l'optimisation logique. Si le concepteur veut optimiser son interface de communication, il doit agir sur

la spécification. Donc, le concepteur doit être conscient que la structure de la spécification correspond à l'architecture en écrivant la spécification de l'interface de communication. Comme cette spécification est obtenue en découpant le modèle de port virtuel, cela limite les possibilités pour implémenter indifféremment les éléments de protocole en matériel ou en logiciel.

4.7 Conclusion

Dans ce chapitre, une méthodologie a été décrite pour le raffinement d'une spécification de l'interface en un modèle RTL. A partir de celle-ci, un outil de génération automatique d'interfaces de communication a été développé. Les principes de réalisation de l'outil ont été exposés. Afin de valider la méthodologie, l'outil a été utilisé pour générer quelques interfaces de test. Les expérimentations réalisées sont présentées dans le chapitre suivant.

Chapitre 5
Expérimentations

Sommaire

5.1 Introduction

Ce chapitre présente deux exemples d'application de la méthode de génération des interfaces de communication définie dans le chapitre précédent. Ces expérimentations se limitent à la génération d'interfaces de communication entre un sous-système processeur et un réseau, bien que la méthode s'applique aussi pour des interfaces avec des IP. De telles expérimentations semblent plus pertinentes afin d'évaluer la méthode que la conception d'une interface pour un IP, dans la mesure où le découpage logiciel/matériel est une des perspectives de notre méthode. Il semble également que la génération des interfaces de communication soit plus difficile dans le cas d'interfaces pour des modules logiciels que matériels. La communication avec les IP se fait en accédant aux registres de l'IP avec une relation de type maître/esclave, tandis que des processus logiciels peuvent utiliser des modèles de programmation de haut niveau avec des communications entre deux processeurs qui sont tous les deux des composants maîtres.

L'objectif de la première expérience est la validation de la méthodologie de génération automatique et de l'outil développé. L'utilisation d'éléments de protocole pour modéliser les interfaces de communication est nouvelle, et une des difficultés de la méthode est de maîtriser les concepts du modèle. L'exemple choisi ne comporte pas de difficultés de conception particulières permettant ainsi de se focaliser sur la méthodologie et facilitant sa compréhension. L'évaluation de l'efficacité de la méthode en terme de temps de conception n'a pour l'instant été qu'essentiellement qualitative en raison des difficultés à maîtriser ces nouveaux concepts qui ont requis un travail important

La deuxième expérimentation s'est essentiellement focalisée sur l'intégration et l'utilisation de notre méthodologie pour la conception d'un système. L'analyse du problème du découpage logiciel/matériel des communications présentée au chapitre 3 s'est appuyée sur cette étude de cas.

5.2 Application à la conception d'une interface de communication entre un processeur et des canaux point à point

5.2.1 Présentation de l'expérimentation

L'objectif du premier exemple est d'illustrer simplement les caractéristiques et les principes importants de la méthode. A cette fin, l'exemple choisi est d'une complexité technique limitée mais suffisante pour mettre en valeur l'intérêt de notre méthode. L'exemple montre en quoi le modèle de service facilite la conception des interfaces de communication. L'interface est spécifiée indépendamment de son implémentation. Nous cherchons à montrer que le modèle adopté permet une modélisation rapide et qu'il offre de la flexibilité.

5.2.2 La spécification du système

Notre méthodologie commence par la spécification du système dont nous cherchons à générer les interfaces de communication.

a) L'architecture du système

Le système est composé de trois microprocesseurs. La tâche *T1* s'exécute sur le microprocesseur 1. Elle envoie des données aux tâches *T2.1*, *T2.2* et *T3.1*, *T3.2*. Le microprocesseur 2 exécute les tâches *T2.1* et *T2.2*, et le microprocesseur 3 les tâches *T3.1* et *T3.2*. L'interface logiciel/matériel (pilotes de communication + interfaces de communication) est responsable de router les données de la tâche *T1* vers les quatre autres tâches réceptrices. On veut transmettre les données par paquets de taille fixe. Le réseau de communication est un réseau point à point avec un protocole "handshake". Le réseau est modélisé au niveau transactionnel. On a besoin de deux liaisons point à point pour la communication entre deux tâches s'exécutant sur des processeurs différents. Le protocole de communication utilisé pour le transfert des données nécessite des transferts d'information bidirectionnels. Au total, on a donc besoin de quatre liaisons point à point car les liaisons utilisées sont unidirectionnelles.

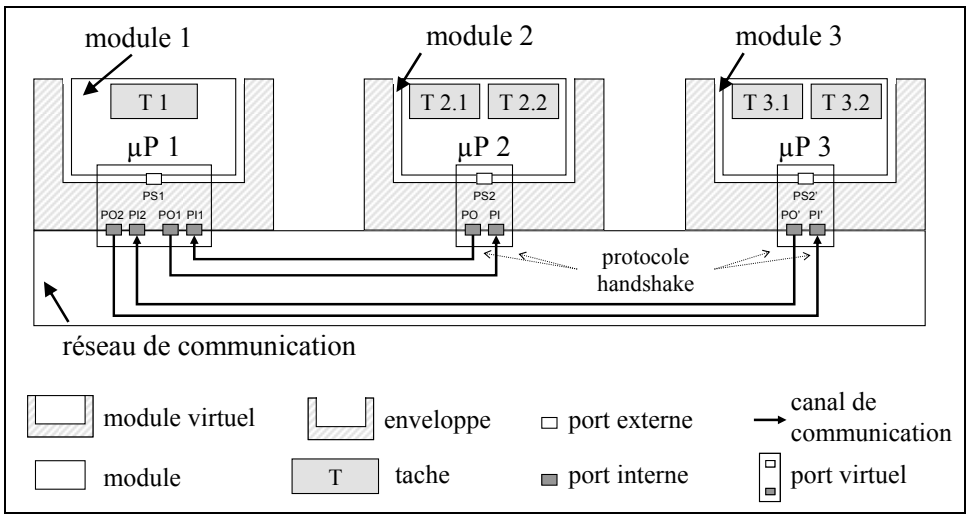


Figure 5.1 - L'architecture virtuelle du système

Notre système est modélisé par une architecture virtuelle (figure 5.1). Avec ce modèle, nos trois processeurs sont représentés par trois modules virtuels. Chaque module virtuel est composé d'un module et de son enveloppe. Le module définit le comportement interne du module virtuel. Dans notre cas, les modules virtuels sont de type logiciel. Les modules sont donc composés des tâches de l'application s'exécutant sur chacun des processeurs. Ainsi, le comportement du module 1 est défini par la tâche T1.

L'enveloppe est définie par un ensemble de ports virtuels. Ceux-ci permettent d'adapter les services de communication demandés à l'application aux ports du réseau de communication. Du point de vue de la génération des interfaces logiciel/matériel, seule la définition des ports virtuels nous intéresse.

Le module 1 possède un port virtuel. Celui-ci possède un port interne (PS1). Les ports internes sont les ports d'accès aux services de communication de l'interface. On compte également quatre ports externes sur le module 1 (PO1, PO2, PI1, PI2). Ce sont les ports du réseau de communication.

Le module 1 possède deux ports de sortie et deux ports d'entrées. Les modules 2 et 3 possèdent chacun un port d'entrée et un port de sortie. Le routage des paquets vers le module 2 ou 3 fait partie de la spécification du protocole. C'est une donnée du problème. On représente donc l'interface logiciel/matériel du module 1 avec un seul port virtuel car on a un service unique.

Du point de vue de la génération des interfaces de communication, le problème peut-être modélisé ainsi :

MODULE 1 :

Protocole de communication : *DEMUX_CHECKSUM_ACK*

Domaine d'horloge : CLK1

Définition des ports internes :

- **PS1** :
  - Service requis : *PA:RE(paquet, id_tâche)* -> émettre un paquet de données à la tâche *id_tâche*
  - FORMAT DES DONNEES : *paquet(taille=8, mot=32 bits)* -> paquet de taille fixe de mots de 32 bits. La taille des paquets est fixée mais paramétrable.
    paramètres :
    - taille paquet = 8
    - taille mot = 32
  - API : *send_paquet(id_process, paquet)*

Définition des ports externes :

- **PO1** : port *out*, protocole *HNDSHK*, taille_donnée : *32 bits, master*, domaine d'horloge : *asynch*
- **PO2** : port *out*, protocole *HNDSHK*, taille_donnée : *32 bits, master*, domaine d'horloge : *asynch*
- **PI1** : port *in*, protocole *HNDSHK*, taille_donnée : 3 *bits, slave*, domaine d'horloge : *asynch*
- **PI2** : port *in*, protocole *HNDSHK*, taille_donnée : 3 *bits, slave*, domaine d'horloge : *asynch*

MODULE 2 :

Protocole de communication : *DEMUX_CHECKSUM_ACK*

Domaine d'horloge : CLK2

Définition des ports internes :

- **PS2** :
  - Service requis : PA:IR(paquet, id_tâche) ->recevoir un paquet de données, le paquet est de taille fixe
  - API : *receive_paquet(id_process, paquet)*

Définition des ports externes :

- **PO** : port *out*, protocole *HNDSHK*, taille_donnée : *3 bits, master*, domaine d'horloge : *asynch*
- **PI** : port *in*, protocole *HNDSHK*, taille_donnée : *32 bits, slave*, domaine d'horloge : *asynch*

MODULE 3 :

Protocole de communication : *DEMUX_CHECKSUM_ACK*

Domaine d'horloge : CLK3

Définition des ports internes :

- **PS2'** :
  - Service requis : PA:IR(paquet, id_tâche, taille) ->recevoir un paquet de données, le paquet est de taille fixe

- API : *receive_paquet(id_process, paquet)*

Définition des ports externes :

- **PO'** : port *out*, protocole *HNDSHK*, taille_donnée : *3 bits, master*, domaine d'horloge : *asynch*
- **PI'** : port *in*, protocole *HNDSHK*, taille_donnée : *32 bits, slave*, domaine d'horloge : *asynch*

Services des canaux de communication :

Les canaux point à point, utilisant le protocole "*handshake"*, offrent un service d'émission d'un mot (W:RE) et requiert le service de réception d'un mot (W:IR).

Cette description représente la spécification fournie en entrée par l'utilisateur.

b) Le modèle des communications

Quand le système est modélisé avec une architecture virtuelle, le concepteur doit choisir son protocole de communication. Dans ce but, on écrit les modèles des ports virtuels en assemblant des éléments de protocole.

Le protocole de communication

On a défini pour cela le protocole *DEMUX_CHECKSUM_ACK* pour la transmission de paquets de données entre un émetteur et deux récepteurs. L'intégrité des données est assurée. On utilise un contrôle par checksum pour détecter les erreurs de transmission. L'émetteur additionne l'ensemble des données à transmettre. L'addition se fait sur n bits (avec n le nombre de bits des données), on calcule donc la somme modulo $2^n - 1$. On obtient ensuite le checksum en réalisant le complément à 1 de cette somme. On envoie le checksum à l'émetteur avec les données. Le receveur additionne les données qu'il reçoit avec le checksum émis. Si la somme obtenue est différente de $2^n - 1$ alors il y a eu une erreur de transmission. Le récepteur doit alors demander à l'émetteur de retransmettre le paquet. Sinon le récepteur signale à l'émetteur que les données ont été reçues correctement. Ce mécanisme de transfert est schématisé sur la figure 5.2.

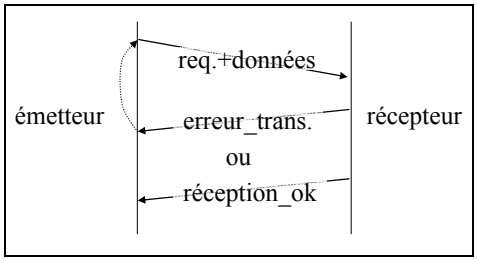


Figure 5.2 - Le schéma des communications

L'acquittement se fait par l'envoi d'une donnée de 3 bits. Si on a une erreur de transmission, alors la donnée renvoyée est égale à 0. Sinon on renvoie une donnée égale à 1. Pour tolérer les erreurs de transmission de la donnée d'acquittement, celle-ci est codée avec un code de Hamming (3,1). On a une

distance de Hamming de 3 qui permet de corriger une erreur. On supporte donc une erreur sur un bit dans l'envoi de la donnée réponse.

Le routage se fait en fonction du numéro de la tâche auquel on souhaite envoyer la donnée. On dispose d'un tableau avec la liste des tâches et le module sur lequel elles s'exécutent.

Modélisation du protocole de communication

Le modèle des communications permet de représenter les interfaces logiciel/matériel. Il décrit ainsi le protocole de communication mais il ne représente qu'un modèle parmi d'autres de celui-ci, car un même protocole peut être représenté par plusieurs modèles différents. De plus, le modèle peut être plus ou moins détaillé selon les fonctions utilisées.

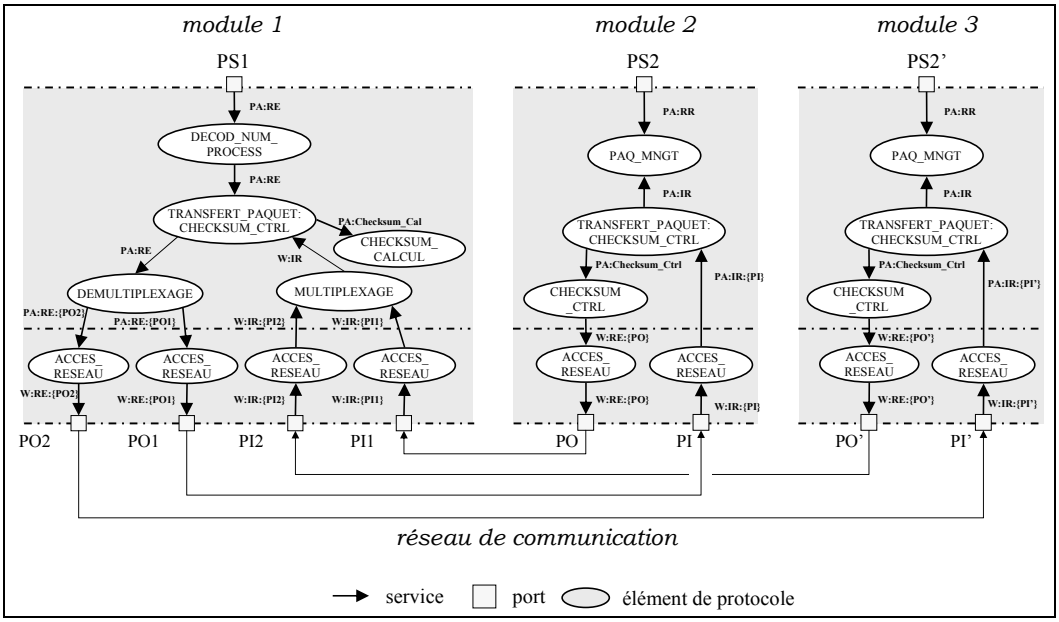


Figure 5.3 - Le protocole de communication

La figure 5.3 représente la spécification des ports virtuels des modules. Comme la tâche T1 veut envoyer des paquets aux modules 2 ou 3, ce besoin est modélisé par la requête d'un service "envoyer un paquet de données à la tâche *id_tâche*". Ce service est annoté "PA:RE" sur la figure (pour paquet : requête d'émission). Les paramètres de ce service sont l'identité de la tâche cible et les données.

La première opération de notre protocole de communication est de déterminer sur quel module les données doivent être envoyées. Nous l'avons modélisé par l'élément de protocole "DECOD_NUM_PROCESS". Cet élément appelle le service "envoyer un paquet de données au module id_module". Pour la clarté de la figure, on a utilisé sur celle-ci la même notation PA:RE pour représenter tous les services d'envoi de paquets. Une même notation sur la figure peut donc correspondre à deux services différents (PA:RE(paquet, id_tâche) et PA:RE(paquet, id_module) par exemple).

Nous avons choisi un test par checksum pour détecter des erreurs de transmission. L'élément précédent demande pour cela le service "PA:Checksum_Cal" à l'élément de protocole "checksum

calcul". Des éléments de protocole "multiplexage" et "démultiplexage" permettent de guider les données sur le module cible et de recevoir l'acquittement de ce même module.

Les accès au réseau sont représentés par des éléments de protocole "acces_reseau". Un élément de protocole "acces_reseau" est nécessaire pour chaque port du réseau. Les liaisons disponibles sont un canal point à point en entrée et un en sortie avec un protocole "*handshake"* pour chaque module récepteur. On peut modéliser un canal en sortie comme un service "envoyer un mot de donnée à travers le port *nom_du_port*" (W:RE:{*nom_du_port*}) où *nom_du_port* correspond à PO1 ou PO2, et un canal en entrée comme un service "recevoir une donnée à travers le port *nom_du_port*" (W:IR:{*nom_du_port*}) avec *nom_du_port* qui prend les valeurs PI1 ou PI2. Le paramètre de ces services est le mot à envoyer ou à recevoir.

En réception, les modules 2 et 3 requièrent eux le service "Paquet : Requête de Réception". Les données sont reçues par un élément de protocole "accès au réseau" fournissant le service W:IR:{*nom_du_port*}. Cet élément requiert alors le service "PA:IR:{*nom_du_port*}" de l'élément de protocole "TRANSFERT PAQUET: CHECKSUM CTRL" qui s'occupe de contrôler la réception d'un message. Pour cela, elle demande à l'élément de protocole "Checksum Ctrl" le service "PA : Checksum_Ctrl" pour contrôler si le checksum est correct. Si c'est le cas, elle transmet le paquet de données à l'élément de protocole "PAQ_MNGT". Ce dernier élément synchronise la réception des paquets avec la demande de réception de ceux-ci (mémoire tampon au cas où un paquet soit reçu avant que la tâche n'ait besoin de celui-ci).

L'élément de protocole "checksum_ctrl" contrôle si le checksum reçu est correct pour vérifier la validité des données. Si c'est le cas, elle envoie un signal d'acquittement, sinon elle envoie un signal d'erreur. Elle utilise pour cela, le service "W:RE".

c) De la spécification des ports virtuels jusqu'à la spécification de l'interface de communication

Seule l'interface de communication du module 1 a été développée, et nous nous focaliserons donc sur lui dans la suite du document. Après l'écriture de l'architecture virtuelle et du modèle des ports virtuels, le concepteur doit écrire la spécification de l'interface de communication. Il est nécessaire avant de commencer à concevoir les interfaces de communication de découper le modèle des communications. En effet, les interfaces logiciel/matériel sont composées d'une partie logicielle (HAL/pilotes de communication) et d'une partie matérielle (interface de communication). Il est donc nécessaire de répartir la gestion du protocole entre les deux parties et de définir le rôle de chaque partie. La spécification de l'interface est donc composée d'un sous-ensemble du modèle de communication vu au chapitre précédent. A ce sous-ensemble, doit s'ajouter des informations sur la communication entre le logiciel et le matériel.

La première étape est de choisir la liste des éléments de protocole implémentés par l'interface de communication. Dans notre cas, nous voulons être capable de déplacer des tâches d'un module à un

autre ou d'ajouter des tâches. On a ici besoin de flexibilité au niveau de l'allocation des processus sur les processeurs de façon à pouvoir supporter différentes applications ou faire évoluer l'application en rajoutant d'autres processus. L'élément de protocole "DECOD_NUM_PROCESS" du port virtuel est en charge d'identifier sur quel module les données doivent être envoyées selon l'identité de la tâche réceptrice. Le "décodage du numéro de processus" est donc à implémenter de préférence en logiciel pour avoir plus de flexibilité (pas de limitation du nombre de processus, modification de l'allocation des processus). Les autres éléments de protocole sont réalisés en matériel pour optimiser les performances. Le résultat de cette étape est un ensemble d'éléments de protocole liés au réseau de communication.

Ensuite, nous devons choisir le sous-système processeur. La communication ayant maintenant été découpée, il est nécessaire de fixer le protocole de communication entre le logiciel et le matériel : dans quels registres écrit-on ? À quelles adresses ? Il est également nécessaire de fixer les ports de communication entre le logiciel et le matériel. J'entends par là les ports entre la partie matérielle supportant le logiciel et l'interface de communication.

Cela permet de fixer l'interface du sous-système processeur. Le port de l'interface est toujours appelé P_SBSYS. Ensuite, on fixe le protocole utilisé par le pilote pour accéder à l'interface de communication. Le "SAP protocole" du port P_SBSYS est déterminé par l'interface du sous-système processeur et par le protocole utilisé par le pilote.

5.2.3 La spécification de l'interface de communication

La spécification de l'interface est composée de la modélisation de ses fonctionnalités avec un modèle à base d'éléments de protocole et de services (identique à celui utilisé pour le protocole de communication), ainsi que d'un modèle de son interface. C'est-à-dire une définition de ses ports physiques ainsi que des protocoles de communication. La spécification de l'interface n'est donc pas juste un sous-ensemble de la spécification du système.

La figure 5.4 représente la spécification de l'interface de communication. Pour accéder à l'interface de communication, le pilote utilise uniquement les entrées-sorties de l'interface mémoire du processeur ("memory mapped IO"), et l'interface de communication est donc connectée à l'interface mémoire du processeur (un ARM7TDMI dans notre cas). Les ports de cette interface sont représentés par un port appelé P_SBSYS. Il groupe tous les ports du bus processeur.

Le pilote utilise un modèle de programmation pour accéder à l'interface de communication. Pour accéder à l'interface de communication, nous devons premièrement vérifier si elle est disponible en lisant un mot dans un registre à l'adresse ADD_AVBLE. Si le service est disponible, le mot lu est nul, sinon l'interface de communication n'est pas disponible (ex. : elle est encore en train d'envoyer un paquet).

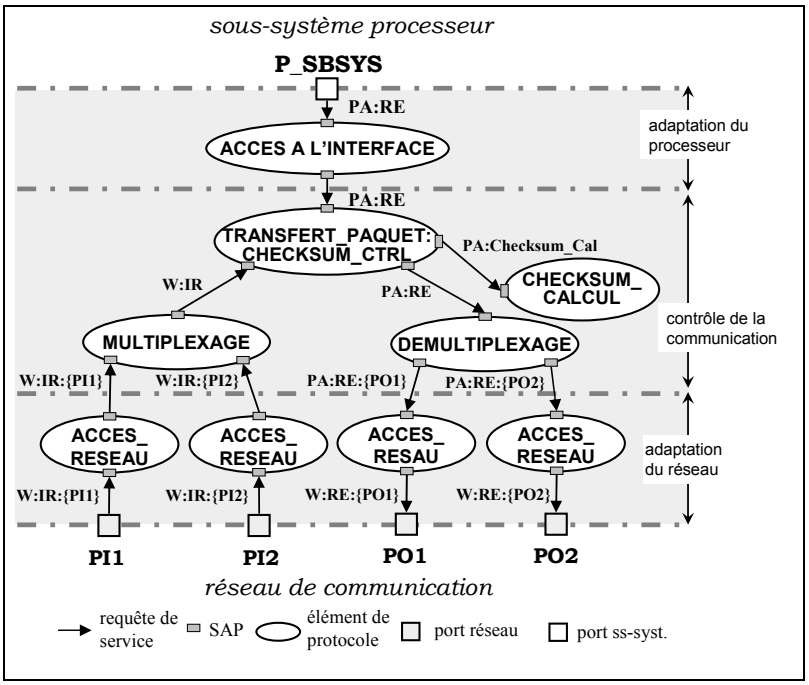


Figure 5.4 - La spécification de l'interface de communication

Si elle est disponible, le processeur doit écrire les données à une adresse particulière. La description du port P_SBSYS et du protocole utilisé par le pilote et le processeur pour accéder à l'interface de communication est contenue dans le paramètre “SAP protocole” attaché à ce port P_SBSYS.

Sur la figure 5.4, on peut voir que l'on ajoute l'élément de protocole “accès à l'interface”. Cet élément de protocole est ajouté systématiquement pour le processus de raffinement. Les services fournis et requis par cet élément sont les mêmes. L'élément de protocole “accès à l'interface” représente l'accès du sous-système processeur à l'interface de communication. Il appelle ensuite le service PA:RE de l'élément de protocole “ TRANSFERT_PAQUET: CHECKSUM_CTRL ” qui est le même que celui décrit précédemment. Les autres éléments de protocole du modèle sont encore les mêmes que ceux décrits dans le modèle du port virtuel.

L'interface de communication accède au réseau de communication par les ports (PO1, PO2, PI1, PI2). Donc, ces ports sont encore présents dans la spécification de l'interface de communication. Un paramètre “SAP protocole” est attaché à ces ports. Cela indique que nous avons des ports avec un protocole *“*handshake*”* à quatre phases. La taille du port est aussi décrite comme un paramètre. Pour envoyer des données, nous utilisons un canal de 32 bits mais pour recevoir l'acquittement nous avons seulement un canal de 3 bits.

L'écriture de la spécification est une tâche pour l'instant manuelle qui doit être faite par le concepteur.

5.2.4 Génération automatique : de la spécification jusqu'à un modèle RTL

Avec notre flot de conception, quatre étapes sont nécessaires pour raffiner la spécification dans un modèle RTL.

a) Modèle des composants de bibliothèque

Pour générer le modèle RTL, un composant de bibliothèque est sélectionné pour tous les éléments de protocole. Le tableau 5.1 donne un exemple de noms et descriptions de composants de bibliothèque qui pourraient être développés et intégrés dans la bibliothèque. Dans notre cas, nous nous sommes limités au développement des composant nécessaires à cette expérimentation, et à la validation du bon fonctionnement de la méthode.

élément de protocole	composant de bibliohèque	description
ACCÈS À L'INTERFACE	contrôleur d'E/S 1	*envoi de paquets entre un ARM7 et deux sous-systèmes*
	contrôleur d'E/S 2	*envoi de paquets entre un MIPS32 M4K et deux sous-systèmes*
	contrôleur d'E/S 3	*envoi d'un mot à partir d'un ARM7*
	...	...
TRANSFERT_PAQUET: CHECKSUM_CTRL	contrôleur de communication 1	*implémentation 1: optimisée en surface*
	contrôleur de communication 2	*implémentation 2: optimisée en fréquence*
	...	...
TRANSFERT_PAQUET: CHECKSUM_CTRL 2	contrôleur de communication 3	*calcul du checksum intégré*
...	...	...

Tableau 5.1 - Liste de composants de bibliothèque

Chaque composant de bibliothèque est décrit en détail. Par exemple, le tableau 5.2 montre les paramètres d'un composant de bibliothèque.

contrôleur de communication							
implemented_PF	PACKET_TRANSFER:CHECKSUM_CTRL						
ports	*nom SAP*	*SAP_Protocole*	*protocole asynchrone*	*RTL ports*			
				nom	*direction*	*type*	*taille*
	SAP_01	PROT_ENBLE_CAL	non	enable	out	std_logic	1
				data	out	std_logic_vector	*PF_parameter(data_size)*
				checksum	in	std_logic_vector	*PF_parameter(data_size)*
	SAP_02	...	...	...	...	...	...
	SAP_03	...	...	...	...	...	...
	SAP_04	...	...	...	...	...	...
	nom			*type*			
	clk			signal d'horloge			
	reset			signal d'initialisation			
paramètres de configuration	*nom*			*valeur*			
	data_size			*PF_parameter(data_size)*			
	packet_size			*PF_parameter(packet_size)*			
fichier HDL	./com_ctrler.vhd.riv						
double horloge	non						

Tableau 5.2 - Paramètres d'un composant de bibliothèque

Sur la première ligne du tableau se trouve le nom du composant. Dessous, on a cinq sections importantes du modèle dans la colonne de gauches : le paramètre implemented_PF, la définition des ports, les paramètres de configuration, le nom du fichier HDL, et un paramètre indiquant si le composant supporte deux horloges. On détaillera l'utilisation et le sens des différentes sections du tableau au cours de chaque étape de la génération (implemented_PF, “double horloge”, ports pendant l'étape de sélection, “paramètres de configuration” pour la configuration, ports pour l'assemblage et “fichier HDL” pour la génération de code).

b) Sélection

La sélection consiste à déterminer toutes les combinaisons valides de composants de bibliothèques implémentant la spécification. Le paramètre "*implemented_PF*" du composant décrit dans le tableau 5.2 indique que ce composant de bibliothèque implémente l'élément de protocole "*PACKET_TRANSFER: CHECKSUM_CTRL*". Dans le tableau 5.2, on peut trouver la correspondance entre les quatre SAP de l'élément de protocole et les ports du composant de bibliothèque. Par exemple, *SAP_01* requiert le service "*PA:Checksum_Cal*". Comme on l'a vu au chapitre 4, le nom des services n'intervient pas dans la génération de l'interface de communication. Les services ne sont donc pas décrits dans le modèle des composants. Seules les relations entre les SAP sont importantes. Le composant RTL utilise le protocole "*PROT_ENBLE_CAL*" pour accéder à ce service (paramètre *SAP protocole* dans le tableau 5.2). Ce protocole utilise trois ports RTL (*enable*, *data* et *checksum)*. Le paramètre *double horloge* indique si le composant supporte deux horloges. Le réseau de communication étant asynchrone, tous les composants sont synchronisés avec l'horloge du processeur.

c) Configuration

L'étape suivante est la configuration des composants RTL. Par exemple, dans le tableau 5.2, la taille du port *data* a la valeur *PF_parameter(data*_size), qui se réfère au paramètre de configuration *data_size* de l'élément de protocole. Les paramètres des éléments de protocole sont assignés lors de l'écriture de la spécification de l'interface. La signification et le type de ces paramètres sont un savoir implicite à la personne utilisant les éléments de protocole. Si la personne qui conçoit les bibliothèques est différente de celle qui écrit la spécification de l'interface, cela implique de documenter chaque élément de protocole sur ses fonctionnalités et ses paramètres de configuration. Dans notre exemple, on doit avoir une documentation indiquant que l'élément de protocole "PACKET_TRANSFER:CHECKSUM_CTRL" calcule le "checksum" de paquets de paquets de *packet_size* mots de données de *data_size* bits, avec *packet_size* et *data_size* deux paramètres de type entier.

d) Assemblage

Un troisième point est la génération de la "netlist" RTL. Chaque élément de protocole est remplacé par le composant RTL sélectionné, et les fils et les bus remplacent les liens entre les services. Les ports physiques avec le même nom sont connectés. Donc, les interconnexions à l'intérieur de l'interface sont uniquement point à point (pas d'interconnexions partagées). Les signaux d'horloge et d'initialisation sont connectés aux composants RTL. Les ports du sous-système processeur et du réseau sont aussi remplacés par un ensemble de ports physiques.

e) Génération de code

Chaque composant RTL a été écrit en VHDL avec du macro-code Rive. Le paramètre "*fichier HDL"* dans le tableau 5.2 est le nom de fichier du modèle HDL.

f) Le modèle RTL

La figure 5.5 représente le modèle RTL de l'interface. Pour des raisons de clarté, les signaux d'horloges et d'initialisation des modules n'ont pas entièrement été représentés sur le schéma mais ils sont définis dans le modèle.

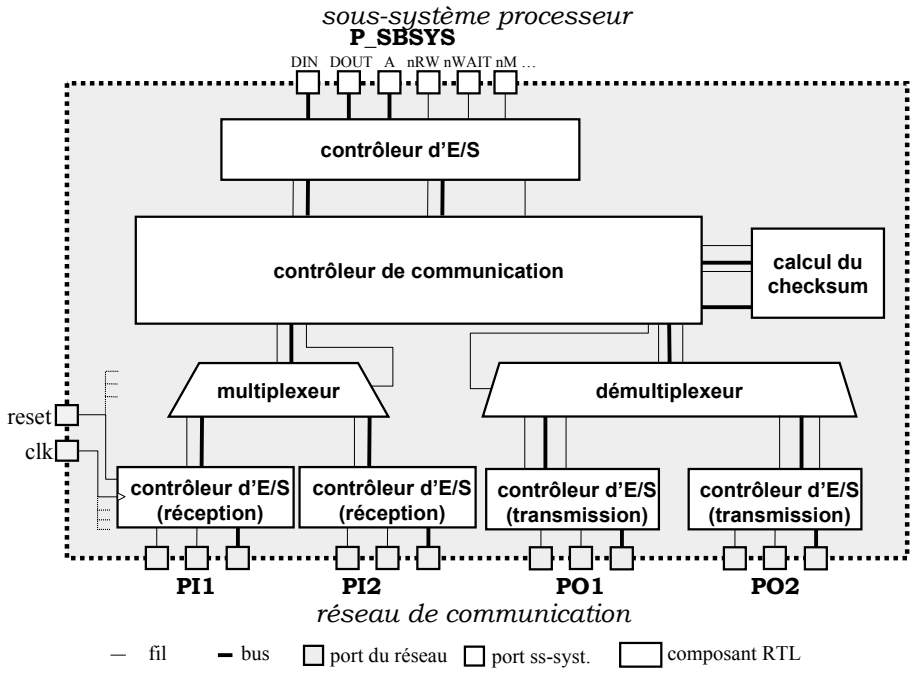


Figure 5.5 - Le modèle RTL de l'interface

Les modules sont définis au niveau RTL. A chaque description d'un module est associé un fichier décrivant en HDL le module. Chaque module possède son code RTL synthétisable. Les ports sont également définis au niveau RTL dans ce modèle. Ainsi à la place du port PO2, on modélise maintenant les ports physiques du canal *"handshake"* avec tous les signaux servant au transfert des données.

5.2.5 Résultats

L'interface a été synthétisée avec la technologie AMS 0.35 µm [AMS] avec Synopsys Design Compiler [Syn01b]. La fréquence de fonctionnement d'un ARM7TDMI sous une technologie 0,35 µm se situant aux alentours des 50 MHz, nous avons fixé comme contrainte pour la synthèse une fréquence minimale de 50 MHz, et optimisé l'interface en surface.

Optimisation de l'interface : différentes implémentations

Pour évaluer notre méthodologie, on a cherché à comparer les résultats obtenus en appliquant notre méthodologie avec les résultats d'une conception manuelle. Le tableau 5.3 montre les résultats de la

synthèse logique pour différents modèles. Les valeurs de surface sont des estimations après synthèse. Nous faisons l'hypothèse que des composants plus petits sont plus réutilisables. Pour estimer la capacité des composants à être réutilisés, on compte donc le nombre de composants utilisés dans la conception. Pour mesurer les performances, on indique la latence de l'interface de communication et le nombre de portes pour évaluer le coût.

La version 1 est l'interface de communication obtenue avec notre méthodologie et la spécification décrite précédemment. Plutôt que de démarrer de rien pour la conception manuelle de l'interface, on a utilisé les composants développés pour la méthode pour concevoir manuellement l'interface. Les versions 2 et 3 de l'interface sont le résultat d'optimisations manuelles de la version 1. La version 2 a été optimisée en latence. La version 3 a été réalisée en décomposant la version 1 en plus petits blocs pour avoir des composants plus réutilisables.

Version	v 1	v 2	v 3
latence (cycles d'horloge)	3	2	3
nombre de composants	9	8	15
nombre de portes (portes équivalentes)	3721	3864	3980

Porte comptabilisé : NAND22 en AMS 0.35 µm

Tableau 5.3 - Résultats

Avec notre méthodologie, l'interface de communication a une latence de trois cycles d'horloge. Comme montré dans le tableau 5.3, il est possible d'obtenir manuellement une latence de deux cycles (version 2). La réduction de la latence a été faite en implémentant l'élément de protocole "checksum_calcul" et " tansfert_paquet: checksum_ctrl" avec seulement un composant RTL. Donc, la latence peut être optimisée manuellement mais on utilise moins de composant RTL. Donc, on limite les possibilités de réutilisation des composants puisqu'ils sont plus spécifiques. La conception de la bibliothèque implique donc de faire des compromis entre les performances et les possibilités de réutilisation des composants. Automatiser la conception de la version 2 pourrait être fait en modifiant la spécification pour regrouper les deux éléments de protocole impliqués. Donc, l'optimisation est encore possible avec notre méthodologie en modifiant manuellement la spécification. La manière de modéliser le port virtuel et de définir les éléments de protocole influence donc les performances de l'interface.

Dans tous les cas, la surface reste relativement constante. Décomposer l'interface de communication en blocs de tailles variables (version 3) a un faible impact sur la taille du circuit. Cela peut être expliqué par le fait que la plus grande partie de l'interface de communication est occupée par la FIFO.

Temps d'écriture de la spécification

Nous avons appliqué notre méthodologie à cet exemple. L'effort de modélisation de la spécification a été faible (moins d'une heure), malgré l'utilisation de fonctions assez sophistiquées (détection d'erreurs par checksum, acquittement et retransmission). L'écriture d'une telle spécification est facile et rapide. Nous avons montré qu'un modèle de spécification basé sur un graphe de dépendances de services peut être raffiné automatiquement en un modèle RTL en utilisant une méthode systématique. De plus, nous pensons que le temps de modélisation de la spécification peut encore être réduit par l'utilisation d'une interface graphique (GUI) puisque le modèle de spécification est bien adapté à une représentation graphique.

Interface spécifique

La génération automatique autorise d'évaluer différentes implémentations alors qu'une exploration manuelle serait trop lente. Un problème pour la réutilisation de composants de bibliothèque est la sélection des composants qui impose d'avoir des composants bien documentés. Le concepteur doit avoir une bonne connaissance des composants pour s'assurer de leurs compatibilités. Nous facilitons la conception d'interfaces en offrant un haut niveau d'abstraction pour l'assemblage de composants. Cela autorise une génération rapide de toutes les implémentations possibles avec les bibliothèques disponibles, pour une exploration systématique de toutes les solutions. L'utilisateur peut optimiser son interface à une application particulière grâce à la facilité de la méthode pour définir l'architecture de l'interface, choisir et assembler les composants. Dans notre cas, l'interface de communication est spécifique à notre exemple qui utilise un réseau de communication et un protocole particulier.

Intégration logiciel/matériel

Evaluer l'intérêt de la méthode pour faciliter l'intégration logiciel/matériel est difficile puisque les pilotes n'ont pour l'instant pas été conçus. Mais la méthodologie nous a offert un format clair pour décrire la frontière entre le logiciel et le matériel. Les fonctionnalités offertes par l'interface au logiciel sont décrites par la définition des services et la manière dont est réalisée la communication entre le logiciel et le matériel est décrite par le SAP protocole.

Nous avons adopté quelques simplifications pour notre réseau, mais nous pouvons malgré tout voir que notre interface de communication est en charge du contrôle des communications.

5.2.6 Conclusion

Nous avons testé notre méthodologie sur un exemple d'application. Les objectifs de cette application étaient à la fois la validation du bon fonctionnement de la méthodologie et de l'outil développé, mais aussi l'estimation de son efficacité. Si cette application a bien permis de valider le bon fonctionnement de l'outil et la possibilité de raffiner un modèle à base de SDG en un modèle RTL, la bonne réutilisation des composants de bibliothèque reste à étudier plus en profondeur. Pour améliorer

la réutilisation des composants, on s'est appuyé sur l'hypothèse qu'il fallait une méthode d'assemblage flexible et des composants de petite taille. On a donc cherché à tester la flexibilité de la méthode et l'influence de la granularité des composants sur le résultat. L'hypothèse initiale n'a par contre pas été validée, et il reste à expérimenter la possibilité de générer une large gamme d'interfaces à partir d'une bibliothèque composée d'un nombre restreint de composants de bases. La définition de ces composants de base fait partie du travail du concepteur de la bibliothèque. La méthode étant flexible, le compromis entre réutilisation et performance des composants n'est pas contraint par la méthode.

5.3 Exemple de réalisation mixte logiciel/matériel de primitives MPI

5.3.1 Présentation de MPI

a) Introduction

Dans une deuxième expérimentation, nous avons développé une réalisation mixte logiciel/matériel de primitives de communications MPI [MPI] sur un système monopuce. Elle a été réalisée en collaboration avec Yanick Paviot dans le cadre de ses travaux de doctorat [Pav04], qui traitent du découpage logiciel/matériel des communications et de la génération des pilotes de communication. La spécification de MPI ("Message Passing Interface") définit un ensemble de primitives standard pour la programmation parallèle à l'aide de processus coopératifs. Il existe deux versions de MPI. MPI-1 vise la communication par passage de messages [MPI-1]. MPI-2 étend la première version pour permettre l'accès à des mémoires distantes, le support des processus dynamiques et des entrées/sorties parallèles [MPI-2].

Même si MPI est un modèle de programmation parallèle initialement destiné aux machines parallèles, les systèmes monopuces sont également des systèmes multiprocesseurs et une application de MPI aux systèmes monopuces a un intérêt pour diminuer les coûts de développement logiciel.

On ne s'est intéressé qu'à MPI-1 qui permet la communication entre des processus logiciels. L'intérêt d'une application de MPI-2 à notre problématique résiderait notamment dans l'ajout de primitives de communication unilatérales ("one-sided communications") destinées à l'accès à des mémoires distantes (RMA ou "Remote Memory Access"). Celles-ci pourraient s'utiliser pour accéder aux mémoires globales ou aux registres des IP des systèmes monopuces, et reste une perspective pour des travaux futurs.

b) Sélection d'un sous-ensemble de primitives MPI

Le standard spécifie les noms, séquences d'appel, résultats et paramètres d'un ensemble de primitives. Toutes les bibliothèques implémentant MPI doivent se conformer au standard pour permettre la portabilité des applications utilisant MPI sur différentes machines. La spécification de MPI définit 125 primitives classées en trois catégories : les primitives de communication, les primitives de dérivation et les primitives de configuration.

Comme les types de données sont prédéfinis par le standard MPI, l'utilisateur définit ses propres structures de données constituées de type prédéfinis en se servant des primitives de dérivation.

Les primitives de configuration servent à la création, la manipulation et la destruction d'objets (un objet MPI est un processus, un groupe de processus ou un "communicator" constitué d'un groupe de processus et d'un contexte de communication).

Les primitives de communication servent à la communication entre les tâches. On peut distinguer les primitives pour les communications multipoints et celles pour les communications point à point. Les primitives de communication multipoints permettent des communications collectives au sein d'un groupe de processus (un transmetteur pour plusieurs récepteurs, plusieurs transmetteurs pour un récepteur, plusieurs transmetteurs pour plusieurs récepteurs).

Les primitives de communication point à point sont bloquantes ou non bloquantes. MPI permet la communication et la synchronisation de processus coopératifs, c'est-à-dire entre deux processus qui demandent explicitement la communication (l'un demande l'envoi de messages et l'autre demande sa réception). Il est donc nécessaire de synchroniser l'envoi et la réception des messages. Le tableau 5.4 regroupe les cinq primitives de communication bloquantes définies par MPI. Il existe quatre modes de communication déclinés dans quatre primitives bloquantes d'envoi de messages, et une unique primitive bloquante pour la réception des messages.

Envoi de messages	Réception de messages
MPI_SEND : envoi standard MPI_BSEND : envoi "avec tampon" MPI_RSEND : "envoi-si-prêt" ("ready Send") MPI_SSEND : envoi synchrone	MPI_RECV : réception standard

Tableau 5.4 - Primitives de communication point à point bloquantes [Pav04]

Dans le cas d'une primitive d'envoi, la tâche exécutant cette primitive est bloquée tant que le message n'a pas été transféré ou copié dans un tampon. Dans le cas de la primitive de réception, la tâche se bloque tant que le message n'a pas été reçu et copié dans le tampon de destination.

Le MPI_SEND peut être appelé indépendamment du moment de l'appel de la requête de réception. La spécification ne définit pas si une mémoire tampon est utilisée, et laisse la décision au concepteur qui implémente cette primitive. Dans certaines implémentations, le système peut éventuellement décider de ne mettre en mémoire tampon que certains messages (typiquement les petits messages pour ne pas remplir le tampon). Si l'utilisateur veut s'assurer qu'une mémoire tampon est ou n'est pas utilisée, il a sa disposition trois autres primitives d'envoi.

La primitive MPI_SSEND permet un envoi synchrone des messages sans utiliser de mémoire tampon. Si la tâche réceptrice est prête, les données sont envoyées. Dans le cas contraire, le processus émetteur est bloqué en attendant la tâche réceptrice. On parle d'envoi synchrone car le transfert se fait en synchronisant la requête d'envoi et celle de réception.

A la différence de MPI_SSEND, la primitive MPI_RSEND ne doit pas être appelée si le récepteur n'est pas prêt, sinon une erreur est générée. L'application doit donc savoir qu'une requête de réception a été postée avant d'appeler un MPI_RSEND.

La primitive MPI_BSEND sert à l'envoi de messages en utilisant une mémoire tampon. Si le tampon est rempli, une erreur est générée.

La primitive MPI_RECV est la primitive standard pour la réception des messages envoyés par les quatre primitives d'envoi de messages bloquantes (MPI_SEND, MPI_BSEND, MPI_RSEND, MPI_SSEND).

MPI définit quatre primitives non bloquantes pour l'envoi de données et une pour la réception de données, correspondant aux versions non bloquantes des primitives précédentes (tableau 5.5). Les primitives non bloquantes peuvent se terminer avant que les requêtes associées n'aient été complètement effectuées et donc que les données associées ne soient réutilisables. Dans le cas d'un envoi, la primitive peut donc se terminer alors que les données à envoyer n'ont encore été ni émises ni copiées dans un tampon. De même dans le cas d'une réception, la primitive se termine alors que les données n'ont pas encore été reçues. La gestion des communications pendantes se fait à l'aide de quatre autres primitives, permettant de connaître l'état des requêtes émises et d'attendre leur achèvement. Les primitives non bloquantes sont intéressantes pour une bonne utilisation du parallélisme entre la communication et le calcul.

Envoi de messages	Réception de messages	Synchronisation explicite et renseignement
MPI_ISEND : envoi standard non bloquant MPI_IBSEND : envoi "avec tampon" non bloquant MPI_IRSEND : "envoi si prêt" non bloquant MPI_ISSEND : envoi synchrone non bloquant	MPI_IRECV : réception standard non bloquant	MPI_WAIT : Attend l'achèvement d'une communication MPI_TEST : Test l'état d'une communication MPI_PROBE : Permet de savoir s'il y a des communications pendantes MPI_CANCEL : Permet de désactiver une communication

Tableau 5.5 - Primitives pour les communications point à point non bloquantes [Pav04]

Un sous-ensemble de primitives MPI de communication point à point adaptées au contexte des SoC a été sélectionné dans [Pav04]. On se limite dans cet exemple au traitement des primitives MPI_SSEND et MPI_RECV, qui sont deux primitives de base de MPI. L'intérêt de ces primitives, outre qu'elles ne requièrent pas de mémoire tampon, est de permettre la synchronisation des tâches. L'envoi synchrone facilite le développement de programme sûr, c'est-à-dire sans blocage ("deadlock"). Avec des envois synchrones, la bonne exécution du programme ne dépend ni du protocole de communication utilisé ni de la taille de la mémoire tampon disponible, ce qui permet une meilleure portabilité. Cela facilite le développement d'une application puisqu'il est possible de développer son programme sur une machine, puis de la porter facilement sur le matériel final.

c) Description détaillée du comportement des primitives sélectionnées

La primitive MPI_SSEND peut être appelée indépendamment de la tâche réceptrice. Comme nous l'avons expliqué précédemment, elle se synchronise avec la primitive de réception. Si la requête de réception du message n'a pas été initiée, la primitive se bloque en attendant cette requête, puis envoie alors le message. Si la requête a déjà eu lieu, les données peuvent être envoyées directement. La syntaxe d'appel de la primitive est la suivante :

MPI_SSEND(buf, count, datatype, dest, tag, comm)

- Buf : adresse du tampon source
- Count : nombre d'éléments constituant le message
- Datatype : type des éléments
- Dest : identificateur du processus destinataire
- Tag : identificateur de message
- Comm : communicator (groupe de processus communicants)

L'appel de la primitive MPI_RECV se fait indépendamment de l'appel de la primitive d'envoi. Si la tâche émettrice n'est pas prête à envoyer le message, le processus est bloqué en attendant l'envoi. La syntaxe d'appel de la primitive est la suivante :

MPI_RECV(buf, count, datatype, dest, tag, comm)

- Buf : adresse du tampon destination
- Count : nombre d'éléments constituant le message à recevoir
- Datatype : type des éléments
- Source : identificateur du processus émetteur
- Tag : identificateur de message
- Comm : communicator

La figure 5.6 montre le graphe des transactions pour le protocole de communication utilisé pour implémenter les MPI_SSEND et MPI_RECV. Les flèches en trait plein représentent des transactions explicites et les flèches en pointillés des événements implicites.

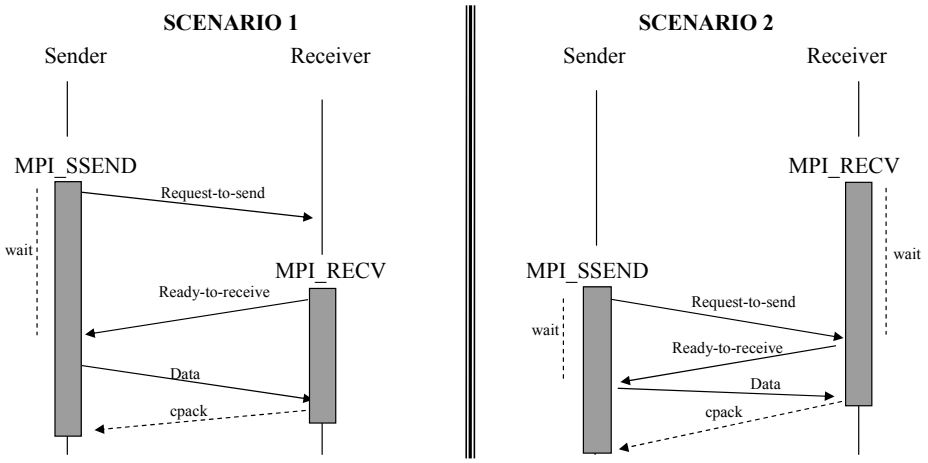


Figure 5.6 - Graphe des transactions

Dans le scénario 1, la requête d'envoi est initiée avant la requête d'émission. A l'appel du MPI_SSEND, l'émetteur transmet au récepteur un "Request-to-send" (contenant un numéro de processus et un identificateur de message), puis attend la réception d'un "Ready-to-receive"

correspondant au message. A la réception du “Request-to-send”, le récepteur vérifie dans une table locale si la requête de réception correspondante a été initiée. Comme ce n’est pas le cas dans le scénario 1, l’information du “Request-to-send” est sauvegardée dans une seconde table locale en attendant la requête de réception correspondante. A l’appel du MPI_RECV, le récepteur sait en lisant dans cette table que la requête d’envoi a été initiée. Il envoie alors un “Ready-to-receive” à l’émetteur pour lui indiquer que le transfert des données peut avoir lieu.

Dans le scénario 2, la requête de réception est antérieure à celle d'émission. L'opération de réception consiste comme précédemment à vérifier dans une table si un “Request-to-send” a été reçu. Comme ce n'est pas le cas, le récepteur attend alors l'arrivée du “Request-to-send”. Une fois la requête d'envoi effectuée et le “Request-to-send” transmis, à son arrivée le récepteur vérifie que la requête de réception correspondante a été émise et sauvegardée dans une table. Il transmet alors en réponse un “Ready-to-receive”, déclenchant l'envoi des données.

L'événement “cpack” représente le fait que les données aient été complètement envoyées au récepteur. Une fois les données transmises, le MPI_SSEND se termine. Le “cpack” est considéré comme un événement implicite car on considère le réseau comme fiable, et une fois les données envoyées on considère qu'elles ont été reçues correctement.

5.3.2 Réalisation mixte logiciel/matériel

a) La spécification du système

La figure 5.7 est une reprise de la figure 3.13 représentant le flot de génération des interfaces de communication défini au chapitre 3.4.2.

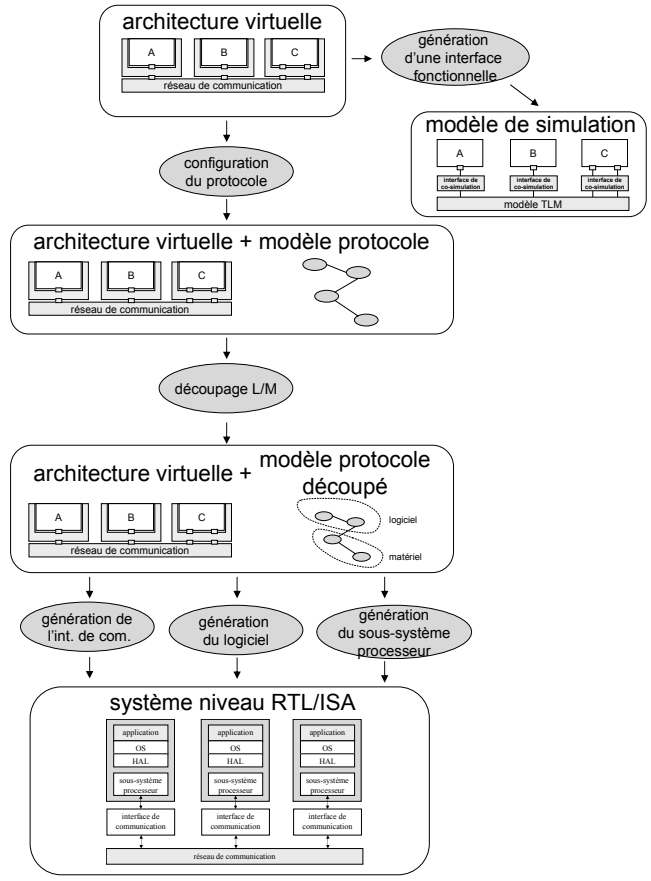


Figure 5.7 - Flot de génération des interfaces de communication défini au chapitre 3.4.2

Comme on peut le voir, le flot commence par la définition de l'architecture virtuelle du système. Notre objectif n'est pas de réaliser une application complète mais de tester l'utilisation de MPI dans les systèmes monopuces. On a donc défini un système multiprocesseur le plus simple possible, puisqu'il est composé uniquement de deux processeurs.

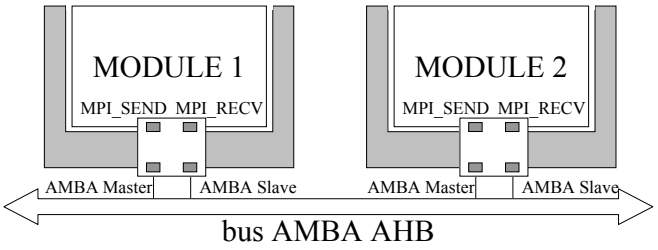


Figure 5.8 - L'architecture virtuelle du système

La figure 5.8 montre l'architecture virtuelle du système ainsi défini. L'architecture se compose de deux modules logiciels : “MODULE 1” et “MODULE 2”. Chacun de ces modules possède un port virtuel offrant à l'application un service d'envoi de messages (MPI_SEND) et de réception de messages (MPI_RECV). Un module peut donc à la fois envoyer et recevoir des messages. Chacun des services requièrent à la fois d'initier et de recevoir des communications. Comme celles-ci sont supportées par un même réseau, les deux services doivent donc être offerts par le même port virtuel.

Le réseau de communication se compose d'un unique bus AMBA AHB 32 bits [ARM99]. Le choix de ce réseau se justifie par son utilisation courante dans des systèmes monopuces commerciaux. On se rapproche ainsi des problématiques industrielles évoquées au chapitre 2.3 en adoptant des hypothèses moins simplificatrices que dans l'exemple précédent où des liaisons point à point avaient été utilisées.

Chaque module est connecté au bus avec un port maître et un esclave, pour être capable à la fois d'initier des transactions et de les recevoir. Le bus AMBA supporte uniquement les transferts pour les communications par MPI.

b) Représentation des primitives MPI sélectionnées et spécification du port virtuel

Notre objectif est une implémentation mixte logiciel/matériel des primitives MPI. Une des premières étapes du flot de la figure 5.7 peut être la réalisation d'un modèle de simulation à partir de l'architecture virtuelle, mais un tel modèle ne présentait pas d'intérêt pour notre problématique. La première étape de notre méthodologie consiste donc à modéliser le protocole (figure 5.7). Ce travail a été fait à la main et fait partie des travaux de [Pav04]. Des choix de conception sont pris à ce niveau. Dans notre cas, nous avons par exemple choisi de découper notre message en paquets. En effet, la communication étant assurée par un bus, le découpage permet de s'assurer que l'envoi d'un message n'utilise pas le bus pendant une trop longue période sans laisser la main à d'autres maîtres. L'utilisation d'un bus AMBA comme réseau de communication garantit que les paquets arrivent dans leur ordre d'émission.

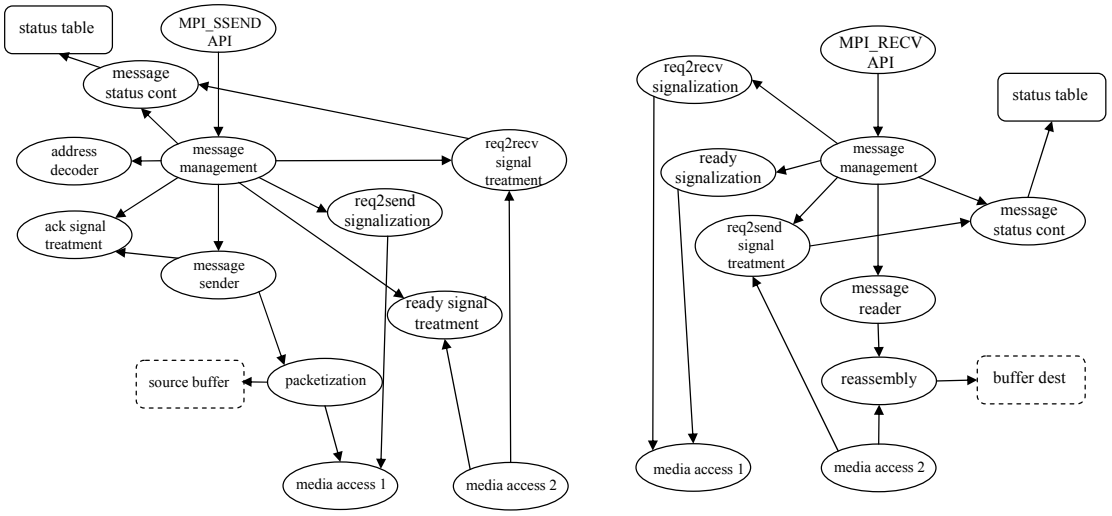


Figure 5.9 - La représentation des primitives MPI sélectionnées

La figure 5.9 montre la modélisation initiale à base d'éléments de protocole des primitives MPI_SSEND et MPI_RECV [Pav04]. Sur la figure 5.9, on peut noter qu'en plus des éléments de protocole (représentés par des ellipses), on trouve des "unités de stockage de données" (représentées par des rectangles). Une unité de stockage de donnée est définie comme étant une entité de mémorisation nécessaire au comportement du service de communication fourni à l'application. Par exemple dans le cas d'une communication utilisant une mémoire tampon (MPI_BSEND), une unité de

stockage de données sert à représenter ce tampon. Un élément de protocole accède à une unité de stockage de données pour la lecture ou l'écriture des données.

Certaines unités de stockage de données sont représentées en pointillés car elles ne font pas vraiment partie du modèle des communications. Ce sont des zones mémoire utilisées et définies par les tâches de l'application, où sont stockées les données à envoyer (tampon source), et où doivent être stockées les données reçues (tampon destination).

Une unité de stockage de données représente une entité de mémorisation, le contrôle de cette entité est assuré par un élément de protocole. On considère dans la suite que cet ensemble est modélisé uniquement par un élément de protocole. Cela permet d'avoir un modèle correspondant au modèle des ports virtuels définis au chapitre 3, avec un seul type d'élément. Les tampons source et destination sont implicites et ne font pas vraiment partie du modèle, ils ne seront également plus représentés dans la suite de cet exemple.

Comme le protocole présenté à la figure 5.6 diffère légèrement de celui présenté dans [Pav04], certains éléments de protocole ont également été supprimés. On obtient alors le modèle de la figure 5.10.

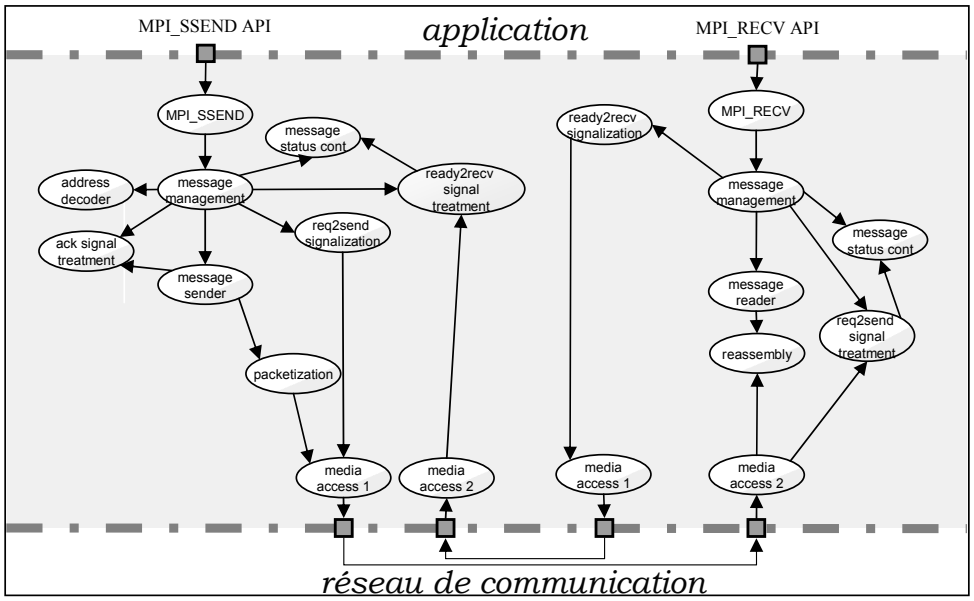


Figure 5.10 - Le protocole de communication

La liste suivante décrit brièvement les éléments de protocole utilisés :

- ***Address Decoder*** : correspondance entre l'adresse où le message doit être envoyé et le numéro de processeur
- ***Message Management*** : gestion de message
- ***Message Sender*** : permet d'envoyer un message
- ***Message Reader*** : permet de recevoir un message
- ***Request to Send Signalization*** : envoi de signal indiquant l'intention d'envoyer un message
- ***Ready to Receive Signalization*** : envoi de signal indiquant l'intention de recevoir un message
- ***Request to Send Signal Treatment*** : traitement des requêtes d'envoi de message
- ***Ready to Receive Signal Treatment*** : traitement des requêtes de réception de message
- ***Ack Signal Send Treatment*** : traitement des signaux d'acquittement
- ***Message Status Controller*** : contrôle un tampon contenant le statut de chaque message
- ***Request Signal Treatment*** : traitement des requêtes

- ***Packetization*** : découpage du message en paquets
- ***Reassembly*** : réassemblage des paquets en messages
- ***Media access 1*** : permet l'envoi de données à travers le réseau
- ***Media access 2*** : permet la réception de données depuis le réseau

La figure 5.9 représente le modèle du protocole utilisé pour les primitives MPI_SEND et MPI_RECV. Comme ces deux primitives utilisent le même réseau, il est nécessaire de regrouper ces deux modèles pour obtenir la spécification du port virtuel (figure 5.11).

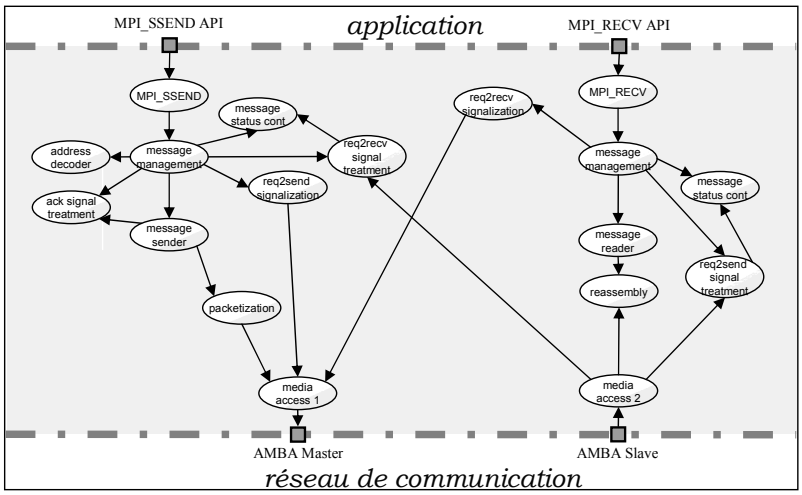


Figure 5.11 - La spécification du port virtuel

c) Découpage logiciel/matériel des communications et choix des sous-systèmes processeurs

Le découpage logiciel/matériel a été réalisé manuellement. Les éléments de protocole responsables de la synchronisation des processus et du contrôle des transferts sont réalisés en logiciel. L'interface de communication s'occupe du découpage en paquets (“packetization”), du réassemblage des messages (“reassembly”) et du transfert des données (“media access 1” et “media access 2”). La figure 5.12 montre les éléments de protocole réalisés en logiciel et ceux réalisés en matériel dans l'interface de communication.

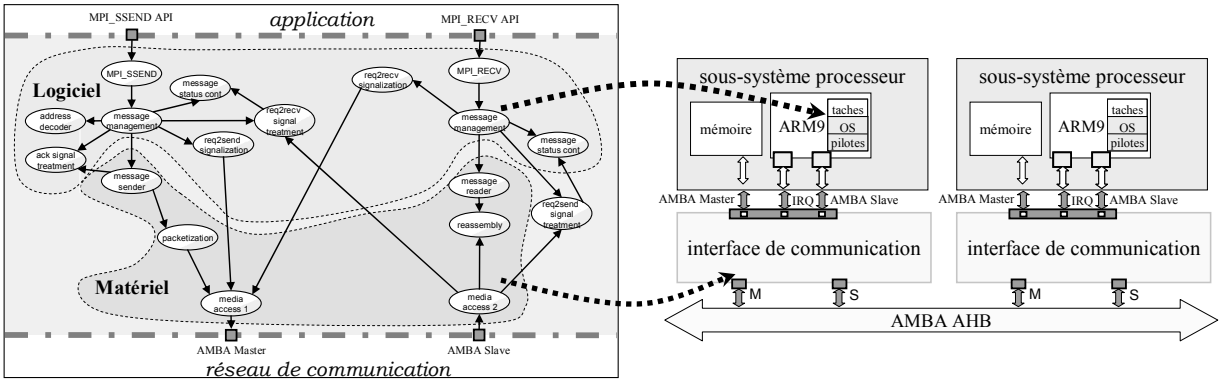


Figure 5.12 - Découpage logiciel/matériel des communications

Lors de l'étape de découpage logiciel/matériel, il est également nécessaire de définir les ports de l'interface de communication et son modèle de programmation pour obtenir la spécification de l'interface de communication. Ceci implique de définir le sous-système processeur.

La figure 5.13 montre un schéma bloc simplifié du sous-système processeur développé par Ivan Petkov au sein du groupe SLS pour la réalisation d'un système d'encodage vidéo DivX. Seuls les composants nécessaires à la compréhension de notre problème ont été représentés mais le lecteur pourra en trouver une description complète dans [Pet05].

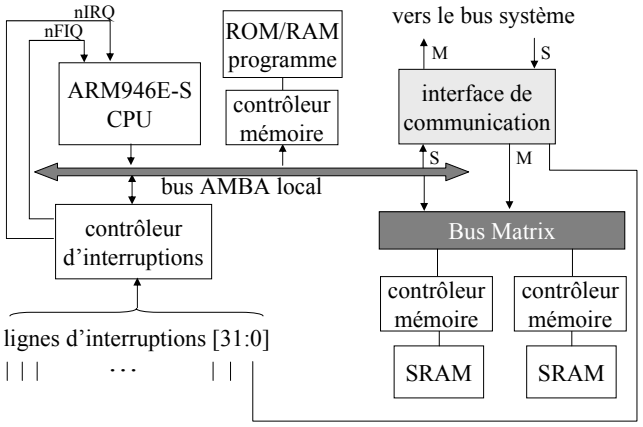


Figure 5.13 - Schéma bloc du sous-système processeur

Le sous-système processeur est construit autour d'un processeur RISC 32 bits ARM946E-S [ARM03]. Une interface avec un bus AMBA AHB 32 bits est intégrée dans le cœur de ce processeur. Un bus AMBA sert donc à connecter un certain nombre de périphériques au processeur. Le sous-système processeur utilise des mémoires locales, le bus du système peut donc être dédié aux transactions des primitives MPI. Un contrôleur d'interruptions est également adjoint au sous-système processeur pour permettre le support de 32 sources d'interruptions. Une de ces lignes d'interruptions est utilisée par l'interface de communication.

Ce sous-système possède également deux mémoires SRAM permettant le parallélisme entre le calcul et la communication. Ces deux mémoires sont connectés à un “bus matrix” [ARM04b] auquel sont également connectés l'interface de communication et le processeur. Un “bus matrix” est un réseau de type “crossbar” constitué de plusieurs bus en parallèle où tous les maîtres sont connectés à tous les esclaves. Dans le sous-système processeur de la figure 5.13, les deux maîtres peuvent donc communiquer avec chacun des deux esclaves (les mémoires). Chaque maître peut accéder à une mémoire différente en parallèle. Le processeur peut donc calculer et sauvegarder des données dans une mémoire, puis demander leur transfert. L'interface de communication peut alors se charger du transfert de ces données en accédant directement à la mémoire. Le processeur peut de son côté continuer à calculer de nouvelles données et à les stocker dans l'autre mémoire sans pénaliser la communication.

On suppose que notre système utilise ce sous-système.

d) Spécification de l'interface

Le découpage logiciel/matériel amène à la spécification de l'interface décrite à la figure 5.14. Elle s'obtient en récupérant les ports du réseau et le sous-ensemble des éléments de protocole de la spécification des ports virtuels à réaliser en matériel, puis par ajout d'un port P_SBSYS et d'un élément de protocole "interface access".

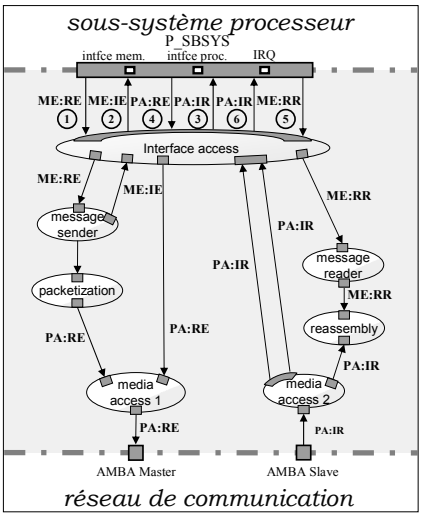


Figure 5.14 - La spécification de l'interface

Quand le protocole a été modélisé, la méthode de génération des interfaces de communication n'avait pas encore été définie et le concept des SAP n'avait pas encore été introduit. Lors de la modélisation de la spécification de l'interface de communication, il a donc fallu définir les SAP des éléments de protocole.

Initialement, un SAP a été défini pour chaque service requis ou fourni, à l'exception des SAP connectés aux ports de l'interface. La conception de l'interface a par la suite montré qu'il était plus intéressant (en terme de taille du circuit) de regrouper deux des services requis par l'élément "media access 2", et la spécification a été modifiée en conséquence.

De plus, les services n'apparaissaient pas de façon explicite dans le modèle dans [Pav04]. La définition de ces services sert uniquement à l'écriture de la spécification des ports virtuels pour l'assemblage des éléments de protocole, et au concepteur des éléments de bibliothèque. La réalisation des composants matériels de la bibliothèque requiert qu'ils soient définis, notamment pour la définition des SAP. Dans le modèle de la figure 5.11, l'élément de protocole "media access 1" fournit un même service à trois éléments de protocole différents. Or, notre méthode de génération des interfaces de communication impose qu'un service requis par un élément de protocole réalisé en matériel ne peut être utilisé par aucun autre élément de protocole. On peut donc alors envisager de modéliser "media access 1" comme un élément de protocole possédant trois SAP fournissant chacun un service identique mais à trois éléments de protocole différents. Mais il faut également tenir compte du fait que plusieurs éléments de protocole réalisés cette fois ci en logiciel peuvent utiliser le même service. L'élément de

protocole “media access 1” a donc été défini comme possédant trois SAP, dont deux fournissent le service “PA:RE” (le troisième est connecté au réseau et requiert du réseau le service PA:RE).

Ce choix est directement lié au découpage logiciel/matériel utilisé. Sur la figure 5.15, on voit que “media access 1” offre le même service (numéroté (4)) à deux éléments de protocole implémentés en logiciel.

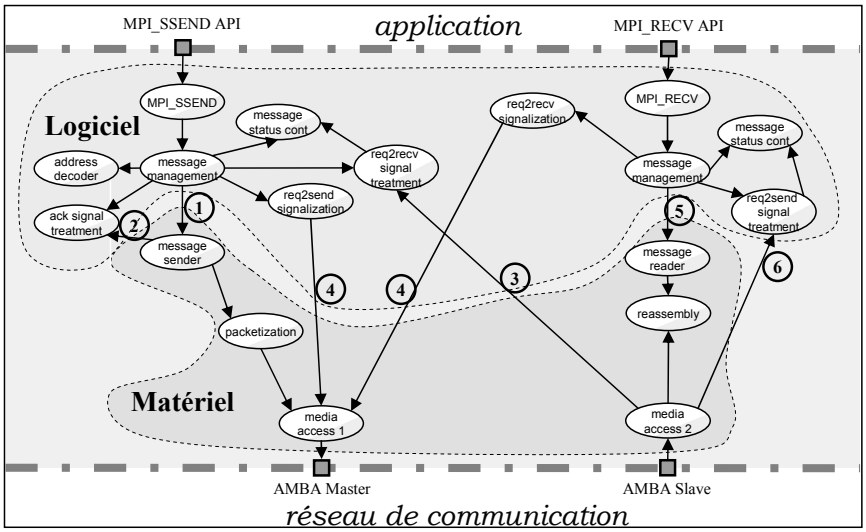
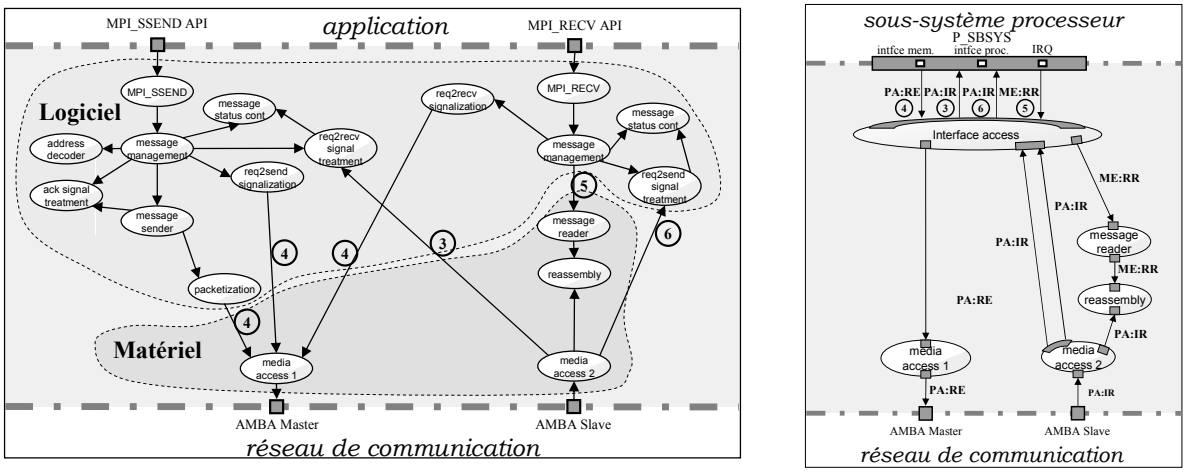


Figure 5.15 - Services fournies par l'interface

Ce service peut être fourni par le même SAP de “media access 1”. Si on avait décidé de réaliser aussi l'élément de protocole “packetization” en logiciel, un unique SAP de “media access 1” aurait pu fournir ce service aux trois éléments l'appelant. Le découpage serait alors devenu celui de la figure 5.16.a) et la spécification de l'interface correspond à la figure 5.16.b).



a) b)

Figure 5.16 - illustration des influences du découpage logiciel/matériel

La façon de modéliser la spécification des ports virtuels n'est donc pas complètement indépendante des choix de réalisation logiciel/matériel.

e) Modèle de programmation de l'interface de communication

Le paramètre “SAP_protocole” attaché au port P_SBSYS définit les ports physiques connectant l'interface avec le sous-système processeur (un port AHB maître, un port AHB esclave et une ligne d'interruption). Ce paramètre définit aussi le modèle de programmation de cette interface. Pour chaque service de l'interface accédé par le logiciel, il est nécessaire de définir les méthodes permettant d'utiliser ce service. On peut voir sur la figure 5.15 que l'interface de communication fournit six services au logiciel.

Par exemple, le modèle de programmation permettant d'accéder aux services (1) et (2) est le suivant :

- service (1) :
  - scrutation du registre d'adresse *ADDR_DISPO_MS* en attente d'une valeur nulle indiquant la disponibilité du service
  - écriture à l'adresse *ADDR_MSSG_SD* de l'adresse destination, du nombre de mots à transférer et de l'adresse source
- service (2) :
  - déclenchement d'une interruption
  - lecture du numéro d'interruption à l'adresse *ADDR_IRQ,* ce numéro d'interruption est égal à 3

Les différentes adresses sont définies dans la spécification de l'interface comme des paramètres de configuration de l'élément de protocole “interface access”. On voit donc que beaucoup d'informations sont contenues dans un même paramètre, puisque le paramètre SAP_protocole définit à la fois des ports physiques et un modèle de programmation. Il faut prêter une grande attention lors de la définition de ce paramètre car c'est une étape compliquée et source d'erreurs. A cause de ce regroupement de nombreuses informations dans un seul paramètre, il semble également qu'il est difficile de réutiliser l'élément de protocole “interface access” car il est alors très spécifique à l'application. Cette limitation est intrinsèque au modèle utilisé, et il faudrait faire évoluer le modèle pour y remédier.

5.3.3 L'interface de communication

La dernière étape de la méthodologie nous a permis d'obtenir l'interface de communication au niveau RTL (figure 5.17). Les signaux d'horloge et d'initialisation ne sont pas représentés pour la clarté de la figure.

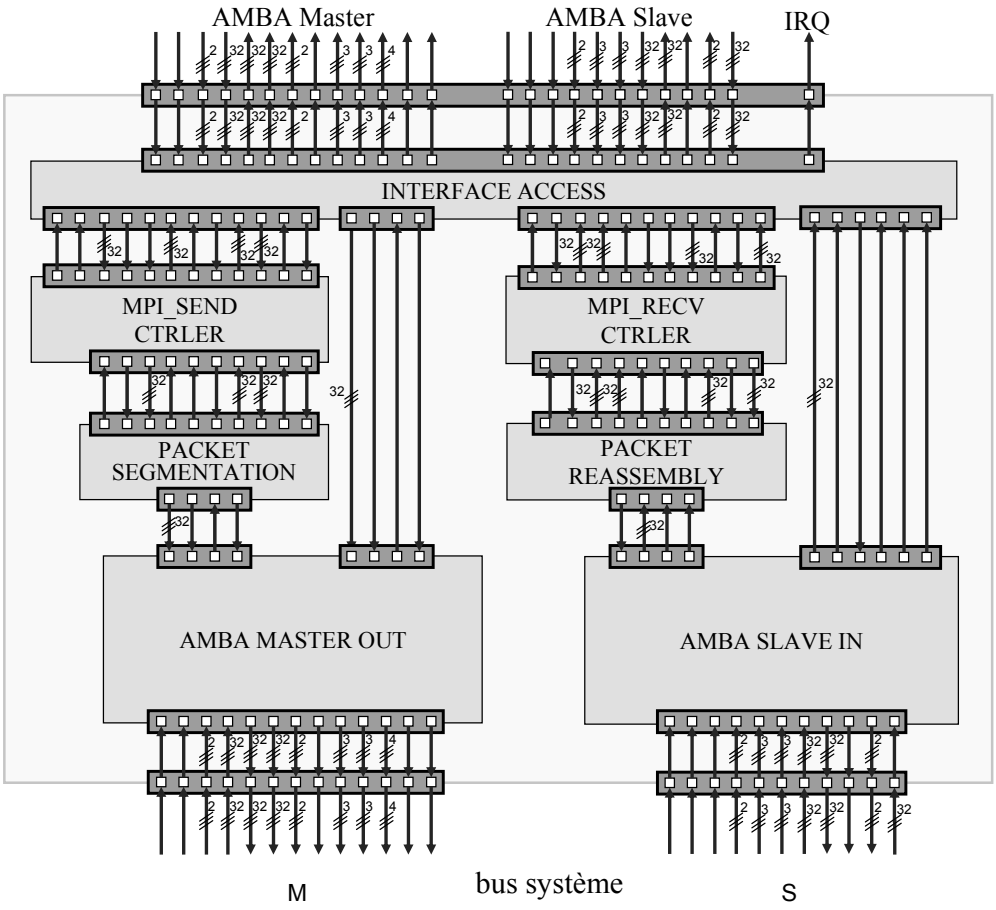


Figure 5.17 - Modèle RTL de l'interface de communication

Les composants ont été écrits en VHDL (avec des macro pour la configuration). Au jour d'aujourd'hui, ces composants n'ont pas encore été intégrés dans la bibliothèque et il reste à écrire la spécification du système et des ports virtuels en Vadel. La méthodologie proposée a donc été appliquée manuellement dans le cadre de cette expérience. La synthèse logique s'est faite avec une technologie AMS 0.35 µm [AMS]. Le tableau 5.6 montre les résultats obtenus.

nombre de portes	surface (mm²)	fréquence maximale (MHz)
13728	0,75	84

Tableau 5.6 - Résultats de la synthèse logique

Les fréquences maximales de fonctionnement d'un processeur ARM9 à notre disposition sont pour des technologies 0.13 µm et 0.18 µm. On ne tire donc aucune conclusion par rapport à la fréquence obtenue. On a également à notre disposition les tailles d'un processeur ARM9 dans des technologies 0,13 ou 0,18 µm. On a par contre adopté ici l'hypothèse que le passage d'une finesse de gravure de 0,35 à 0,13 µm permettait une diminution de la surface du circuit d'un même rapport dans les deux dimensions. Cette estimation grossière nous donne une taille de l'interface avec une technologie 0,13

µm de $\left(\frac{0.13}{0.35}\right)^2 * 0.7 \approx 0.1\,\text{mm}^2$. La taille d'un ARM946E avec une technologie 0,13 µm est de 1,96 mm² sans cache, et 3,26 mm² avec un cache de 2*8 kB [ARM]. Même si la méthode d'estimation de la taille du circuit est approximative et sujette à une marge d'erreur importante, on a donc un facteur de 20~30 entre la surface de l'interface et celle du processeur. La taille du circuit est donc relativement faible.

Cette expérimentation a montré que la méthodologie proposée dans cette thèse pour la conception des interfaces permettait de réaliser des interfaces relativement complexes. Elle ouvre la voie également à quelques perspectives. Il serait notamment intéressant d'essayer d'autres découpages logiciel/matériel et de voir l'influence de ces découpages sur la taille de l'interface et les performances des communications. Une seconde perspective est l'implémentation d'autres primitives MPI tel qu'un MPI_SEND avec mémoire tampon afin de mesurer le nombre de composants matériels pouvant être réutilisés d'une implémentation à une autre.

5.3.4 Conclusion de l'exemple

Un outil de génération automatique des interfaces de communication a été développé au cours de cette thèse. La méthodologie utilisée repose sur l'utilisation d'un modèle à base d'éléments de protocole comme spécification de l'interface. Cet exemple a cherché à tester si cette méthodologie répondait aux objectifs initiaux qui étaient l'optimisation des communications et la réalisation d'une interface de communication optimale. On a vu au cours de ce chapitre que le découpage logiciel/matériel n'est pas évident même en utilisant un modèle du protocole de haut niveau. De plus, la méthodologie de génération des interfaces de communication impose certaines contraintes qui vont limiter les choix de découpage et compliquer la modélisation du protocole. Cette modélisation étant loin d'être triviale, l'automatisation d'un flot tel que celui qui a été présenté dans le chapitre 3 est une perspective très importante pour pleinement exploiter la méthodologie proposée, même si de nombreux problèmes restent à résoudre pour le développement d'un flot complet.

Chapitre 6
Conclusion et perspectives

Sommaire

6.1 Problématique de la communication dans les systèmes monopuces

L'intégration de processeurs a profondément modifié les circuits intégrés avec l'arrivée des systèmes monopuces et a ainsi bouleversé les méthodes de conception. Les délais et les coûts de conception des systèmes monopuces sont dus au manque de méthodologie et d'outil pour guider et assister le concepteur. La communication dans les systèmes monopuces est un des problèmes clés de leur conception. Des flots de synthèse au niveau système ont été proposés, et reposent sur un découplage du traitement des données et de la communication. La spécification du système est un modèle où les communications sont modélisées à un haut niveau d'abstraction afin d'en maîtriser la complexité. Cette thèse traite de la génération automatique des interfaces de communication, afin d'accélérer le raffinement de ces modèles des communications.

6.2 Revue des travaux présentés

6.2.1 Conception des interfaces de communication

Le chapitre 2 est parti d'une étude de cas pour expliquer la complexité des communications et l'intérêt de les optimiser pour une application spécifique. Ces optimisations sont généralement faites manuellement, et imposent le développement d'interfaces de communication spécifiques. L'utilisation de modèles en couches issus du monde des réseaux est un moyen pour structurer l'abstraction des communications pouvant être appliqué au contexte des SoC. Lors de la conception finale du circuit, cette structure ne peut être gardée, et il est nécessaire d'utiliser des interfaces de communication. Une étude des méthodes existantes pour la génération des interfaces de communication a donc été présentée dans ce chapitre. Elle a permis de fixer les directions de recherche des travaux menés pendant cette thèse. On a cherché à définir une approche commune pour la génération des interfaces et des pilotes de communication, afin de pouvoir envisager différents choix de réalisation logiciel/matériel. De plus, on s'est fixé comme contrainte d'avoir une méthode flexible et partant d'une spécification avec un haut niveau d'abstraction permettant ainsi d'optimiser facilement l'interface pour un système spécifique.

6.2.2 Modélisation et raffinement des communications

Le chapitre 3 a présenté un modèle des communications, qui se veut une alternative aux modèles en couches. Il représente les communications par un graphe de dépendance de services. L'utilisation de ce modèle vise à fournir une solution au problème de l'optimisation et de la configuration des communications. Un flot pour raffiner ce modèle en une réalisation logiciel/matériel des communications a été discuté.

6.2.3 Un outil de génération automatique des interfaces de communication

Dans le chapitre 4, une méthodologie pour la génération automatique des interfaces de communication a été proposée. Celle-ci repose sur un mécanisme d'assemblage de composants de bibliothèques. Un outil de génération a été développé à partir de cette méthodologie.

6.2.4 Expérimentations de la méthodologie

Les deux expérimentations présentées dans le chapitre 5 avaient pour but de tester et d'évaluer la méthodologie proposée.

L'application de notre méthodologie à un premier exemple de système a permis de valider le bon fonctionnement de la méthode. Cette première expérimentation est relativement simple, ce qui lui permet de bien illustrer et de faciliter la compréhension de la méthode. Elle a néanmoins fait apparaître la principale limite de notre méthodologie qui est l'utilisation de bibliothèques.

La deuxième expérimentation présentée est une implémentation mixte logiciel/matériel de primitives de programmation parallèle MPI pour les systèmes monopuces. Elle a mis en valeur les difficultés liées à la modélisation d'un protocole de communication avec un graphe de dépendance de services et les limites concernant le découpage logiciel/matériel des communications à partir de ce graphe. Malgré ces difficultés, elle a montré que l'on pouvait générer des interfaces de communication complexes, afin de supporter partiellement en matériel des primitives de communication de haut niveau.

6.3 Perspectives

6.3.1 Configuration de la spécification des ports virtuels

La spécification des ports virtuels par un graphe de dépendance de services laisse entrevoir de générer automatiquement cette spécification. La génération de ce graphe se ferait en sélectionnant certains éléments de protocole à partir des primitives de l'application et du réseau de communication modélisé dans l'architecture virtuelle. Il est également envisageable d'attacher certains paramètres à l'architecture virtuelle pour guider cette sélection.

6.3.2 Découpage logiciel/matériel et estimation de performances

Le modèle des communications qui a été défini et le flot de raffinement associé ont pour objectifs de permettre l'optimisation des protocoles de communication et la réalisation d'implémentations mixtes logiciel/matériel optimales. L'automatisation de la génération des interfaces de communication est nécessaire pour l'efficacité de ce flot mais n'est pas suffisant. Il est également important d'automatiser l'exploration des différentes solutions de réalisation logiciel/matériel, et de guider cette exploration par des contraintes de coût/performance. Cela implique de pouvoir évaluer les différentes solutions possibles rapidement. Une simulation au niveau RTL étant relativement lente, le

développement de modèles d'estimation de performances est une perspective de ce travail. Comme le modèle des communications proposé peut être utilisé pour générer à la fois le logiciel et l'interface de communication, cela laisse entrevoir la perspective d'un tel modèle d'estimation de performances.

Une voie possible à ce problème pourrait être de modéliser les services et les éléments de protocole sous forme de réseau de Petri temporisé. L'utilisation de réseau de Petri pour la modélisation des communications dans les SoC a déjà été traitée mais elle requiert d'écrire manuellement un réseau de Petri pour chaque système. Chaque composant logiciel ou matériel réalisant un élément de protocole serait modélisé par des places et des transitions. La composition des éléments de protocole guiderait la composition des différents modèles de performances des éléments pour construire un réseau de Petri temporisé, modélisant les performances de l'interface logiciel/matériel.

6.3.3 Modélisation du sous-système processeur par un graphe de dépendances de services

Le choix d'une réalisation logiciel ou matériel d'un élément de protocole dépend du sous-système processeur sur lequel le logiciel va tourner. Il pourrait donc être nécessaire d'avoir un modèle du sous-système processeur pour pouvoir automatiser le découpage logiciel/matériel. La modélisation d'un sous-système processeur par un graphe de dépendance de services est une perspective à étudier. Le sous-système processeur pourrait être alors configurer à partir de ce graphe de dépendance de services.

De plus, dans toutes nos expérimentations, les communications étaient toujours initiées explicitement par le logiciel. Nous n'avons pas envisagé les cas où c'est le sous-système processeur qui initie la communication (par exemple un contrôleur de mémoire cache qui accède au réseau pour mettre à jour la mémoire cache). La spécification de l'interface devrait alors être modifiée en fonction du sous-système processeur.

6.3.4 Synthèse des interfaces de communication

Un des problèmes identifiés de notre méthodologie de génération automatique est l'utilisation de bibliothèques de composants. La réutilisation des composants de bibliothèque est donc un point extrêmement important. Pourtant, la réutilisation des composants ne semble pas complètement optimale. Les composants servant à la réalisation des éléments de protocole “accès à l'interface” pourraient notamment être décomposés en blocs de plus faible granularité. Pour réutiliser ces composants, il faudrait utiliser un autre mécanisme d'assemblage. Ce mécanisme nécessiterait sûrement un modèle des SAP_protocole. Un tel mécanisme d'assemblage nécessiterait par exemple de connaître le nombre de types d'interruptions différents pour générer le contrôleur d'interruptions adéquat.

6.3.5 Modélisation des composants avec différents langages

Le langage HDL utilisé pour modéliser les composants n'est pas contraint par l'outil. Certains éléments fortement “orientés contrôle” peuvent donc être utilisés avec des langages formels comme ceux décrits au chapitre 2.5.3.

6.3.6 Expérimentation de la méthodologie avec des IP matériels

Les expérimentations réalisées ont porté sur la génération d'interfaces de communication entre un processeur de communication et un réseau de communication. Cela permettait en effet d'étudier l'intérêt de la méthode pour le découpage logiciel/matériel des communications. La méthodologie employée est théoriquement aussi applicable pour connecter des IP à un réseau, et il serait intéressant de l'expérimenter.

Bibliographie

[Alm89] G. S. Almasi and A. Gottlieb, “Highly parallel computing”, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989.

[AMS] austriamicrosystems, http://www.austriamicrosystems.com/index.htm.

[ARM] ARM, http://www.arm.com.

[ARM99] ARM Limited, “AMBA Specification (Rev 2.0)”, 1999.

[ARM01] ARM Limited, “AHB CPU Wrappers Technical Reference Manual”, 2001.

[ARM02] ARM Limited, “ARM966E-S (Rev2) Technical Reference Manual”, 2002.

[ARM03] ARM Limited, “ARM946E-S Technical Reference Manual”, version r1p1, 15 mai 2003. Disponible sur : www.arm.com.

[ARM04a] ARM Limited, “ARM926EJ-S (r0p4/r0p5) Technical Reference Manual”, 26 janvier 2004. Disponible sur: www.arm.com.

[ARM04b] ARM Limited, “Multi-Layer AHB Overview”, mai 2004. Disponible sur : www.arm.com.

[Arteris] ARTERIS, http://www.arteris.com.

[Bag98] A. Baganne, J. L. Philippe et E. Martin, “A Formal Technique for Hardware Interface Design”, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 45, n° 5, mai 1998.

[Bea] Beach Solutions, http://www.beachsolutions.com.

[Ben02] L. Benini et Giovanni De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computer, vol. 35, n° 1, p. 70-78, janvier 2002.

[Bho03] P. Bhojwani et R. Mahapatra, “Interfacing Cores with On-chip Packet-Switched Networks”, In Proceedings of the 16th International Conference on VLSI Design (VLSI'03), 2003.

[Bök00] C. Böke, “Combining Two Customization Approaches: Extending the Customization Tool TEReCS for Software Synthesis of Real-Time Execution Platforms”, In Proceedings of the Workshop on Architectures of Embedded Systems (AES2000), Karlsruhe, Allemagne, janvier 2000.

[Bra02] D. Brash, “The ARM Architecture Version 6 (ARMv6)”, white paper, janvier 2002.

[Bru00] J-Y. Brunel, W.M. Kruijtzer, H. J. H. N. Kenter, F. Pétrot, L. Pasqiuer, E. A. De Kock, et W. J. M. Smits, “COSY Communication IP’s”, In Proceedings of Design Automation Conference, juin 2000.

[Bry01] R.E. Bryant, K-T. Cheng, A.B. Kahng, K. Keutzer, W. Maly, R. Newton, L. Pileggi, J.M. Rabaey, A. Sangiovanni-Vincentelli, “Limitations and challenges of computer-aided design technology for CMOS VLSI”, Proceedings of the IEEE, Volume: 89, Issue: 3, p. 341-365, mars 2001.

[Ces01] W. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, A. A. Jerraya, “Colif: A Design Representation for Application-Specific Multiprocessor SOCs”, IEEE Design & Test of Computers, vol. 18, n° 5, septembre/octobre 2001.

[Ces02] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A.A. Jerraya, M. Diaz-Nava, “Component-Based Design Approach for Multicore SoCs”, in Proc. of the DAC, New Orleans, USA, 10-14 juin 2002.

[Che03] E. Chen, Y. Shi, D. Zhang, G. Xu, “A programming framework for service association in ubiquitous computing environments”, Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing and Fourth Pacific Rim Conference on Multimedia (ICICS-PCM 2003), 2003.

[Cul99a] D. E. Culler et J. Pal Singh, avec Anoop Gupta, “Parallel Computer Architecture”, chapitre 1, Morgan Kaufmann Publishers, 1999.

[Cul99b] D.E. Culler, J. Pal Singh, “Parallel Computer Architecture”, chapitre 7 : Scalable Multiprocessors, Morgan Kaufmann Publishers, 1999.

[Cyr01] G. Cyr, G. Bois, M. Aboulhamid, “Synthesis of communication interface for soc using VSIA recommendations”, Proc. of Designers’ Forum, Design And Test European Conference 2001, Munich, Allemagne, p.155-159, mars 2001.

[D&R] Design and Reuse, http://www.us.design-reuse.com.

[Dal01] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, in Proceedings of the Design Automation Conference, p. 684–689, Las Vegas, juin 2001.

[Die03] J. Dielissen, A. Radulescu, K. Goossens, et E. Rijpkema, “Concepts and Implementation of the Philips Network-on-Chip”, IP-Based SOC Design, Grenoble, novembre 2003.

[Döm98] R. Dömer, D. D. Gajski, J. Zhu, “Specification and Design of Embedded Systems”, it+ti magazine, Oldenbourg Verlag, Munich, Allemagne, n° 3, juin 1998.

[Don04] A. Donlin, “Transaction level modeling: flows and use models”, Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS'04), Stockholm, Suède, p. 75 – 80, septembre 2004.

[Dut01] S. Dutta, R. Jensen, and A. Rieckman, “Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems”, Septembre-Octobre 2001.

[Dzi04] Mohamed-Anouar Dziri, “Modèles d'intégration d'outils et de composants logiciel/matériel pour la conception des systèmes hétérogènes embarqués”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, mai 2004.

[Eij99] J.T.J. van Eijndhoven, F.W. Sijstermans, K.A. Vissers, E.-J. Pol, M.J.A. Tromp, P. Struik, R.H.J. Bloks, P. van der Wolf, A.D. Pimentel, and H.P.E. Vranken, “TriMedia CPU64 Architecture”, in Proceedings of International Conference on Computer Design (ICCD), Austin, Texas, p. 586-592, octobre 1999.

[Ens01] C. Ensel, “A Scalable Approach to Automated Service Dependency Modeling in Heterogeneous Environments”, 5th International Enterprise Distributed Object Computing Conference (EDOC '01), 2001.

[Fly97] D. Flynn, “AMBA: Enabling Reusable On-Chip Designs”, IEEE Micro, 1997.

[Gau01] Lovic Gauthier, “Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, mai 2001.

[Gha03] Ferid Gharsalli, “Conception des interfaces logiciel-matériel pour l'intégration des mémoires globales dans les systèmes monopuces”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, juillet 2003.

[Goo04] K. Goossens, O. P. Gangwal, J. Roever, A. P. Niranjan, “Interconnect and Memory Organization in SOCs for advanced Set-Top Boxes and TV --- Evolution, Analysis, and Trends”, In “Interconnect-Centric Design for Advanced SoC and NoC”, Jari Nurmi, Hannu Tenhunen, Jouni Isoaho, Axel Jantsch, editors. Kluwer, avril, 2004.

[Gue00] P. Guerrier, A. Greiner, “A Generic Architecture for On-chip Packet-switched Interconnections”, Proceedings of the DATE'2000 Conference, Paris, France, p. 250-256, mars 2000.

[Hav02] A. Haverinen, M. Leclercq, N. Weyrich, D. Wingard, “SystemC based SoC Communication Modeling for the OCP Protocol”, white paper, octobre 2002. Disponible sur: http://www.ocpip.org.

[Hom01a] D. Hommais, F. Pétrot and I. Augé, “A Practical Tool box for System Level Communication Synthesis”, in Proc. IEEE International Workshop on Rapid System Prototyping, 2001.

[Hom1b] D. Hommais, “Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel”, Thèse de Doctorat, Paris VI, 2001.

[IBM99] IBM, “The CoreConnect™ Bus Architecture”, 1999, white paper. Disponible sur : http://www-3.ibm.com/chips/products/coreconnect.

[ITRS] International Technology Roadmap for Semiconductors, “International Technology Roadmap for Semiconductors 2001 Edition Front End Processes”. Disponible sur : http://public.itrs.net/Files/2001ITRS/FEP.pdf.

[Jal04] A. Jalabert, S. Murali, L. Benini et G. De Micheli, “×pipesCompiler: A Tool for Instantiating Application Specific Networks on Chip”, DATE 2004, p. 884-889, 2004.

[Jan03] A. Jantsch, “NoCs: A new contract between hardware and software”, in Proceedings of the Euromicro Symposium on Digital System Design, septembre 2003. Invited keynote.

[Jer02] A. A. Jerraya, “De l’ASIC au SoC, puis au réseau de composants sur puce”, Editorial, Veille Technologique, Le Magazine d'ASPROM, n° 25, Dec. 2001/Jan. 2002.

[Jer04] A. A. Jerraya, “HW-SW Interfaces Modeling & Design for MPSoC”, Revue des travaux du Groupe de Synthèse au niveau Système, Seyssins, janvier 2004.

[Kar01] F. Karim, A. Nguyen, S. Dey, R. Rao, “On-chip Communication Architecture for OC-768 Network Processors”, Proceedings of the 38th conference on Design automation, juin 2001.

[Lah00] K. Lahiri, A. Raghunathan, S. Dey, “Evalutation of the Traffic-Performance Characteristics of System-on-chip Communication Architectures”, in Proc. of the DAC, 2000.

[Len00] C. K. Lennard, P. Schaumont, G. De Jong, A. Haverinen et P. Harde, “Standards for System-Level Design: Practical Reality or Solution in Search of a Question?”, in Proc.of DATE, mars 2000.

[Lin94] B. Lin and S. Vercauteren “Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation”, in Proc. of ICCAD, 1994.

[Lyo01] D. Lyonnard, S. Yoo, A. Baghdadi, A. A. Jerraya “Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System –on-Chip”, in Proc. of DAC, Las Vegas, juin 2001.

[Lyo03] Damien Lyonnard, “Approche d'assemblage systématique d'éléments d'interface pour la génération d'architecture multiprocesseur”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, avril 2003.

[Lys00] R. L. Lysecky, F. Vahid, T. D. Givargis, “Experiments with the peripheral virtual component interface”, proceedings of the 13th international symposium on System synthesis (ISSS), p. 221 – 224, 2000.

[Lys02] R. Lysecky and F. Vahid, “Prefetching for Improved Bus Wrapper Performance in Cores”, ACM Transactions on Design Automation of Electronic Systems, vol. 7, n° 1, p. 58 – 90, janvier 2002.

[Mag02] P. Magarshack, “Improving SoC Design Quality through a Reproducible Design Flow”, IEEE Design & Test of Computers, vol. 19, n° 1, p. 76-83, janvier/février 2002.

[Men03] Mentor Graphics Corporation, “Seamless User's and Reference Manual”, octobre 2002

[Men04] Mentor Graphics Corporation, “Platform Express User Guide”, version 2.1_2.0, mars 2004.

[Mil02] M. Millberg, “The Nostrum Protocol Stack and suggested Services Provided by the Nostrum Backbone”, Draft v0.1.48, novembre 2002, Internal Report.

[Mil04] M. Millberg, E. Nilsson, R. Thid, S. Kumar et A. Jantsch. “The Nostrum backbone - a communication protocol stack for networks on chip”, In Proceedings of the VLSI Design Conference, Mumbai, India, janvier 2004.

[MIPS] MIPS, http://www.mips.com.

[MIPS02] MIPS Technologies Inc., “MIPS SOC-it™ 101 System Controller Family Datasheet”, novembre 2002.

[MPI] Message Passing Interface Forum, http://www.mpi-forum.org.

[MPI-1] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard”, version 1.1, 12 Juin 1995.

[MPI-2] Message Passing Interface Forum, “MPI-2: Extensions to the Message-Passing Interface”, version 2.0, 18 Juillet 1997.

[Nar95] S. Narayan and D. D. Gajski “Interfacing Incompatible Protocols using Inteface Process Generation”, in Proc. of DAC, 1995.

[Nic02] Eugenia Gabriela Nuta Nicolescu, “Spécification et validation des systèmes hétérogènes embarqués”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, novembre 2002.

[Öbe01] J. Öberg, M. O'Nils, A. Jantsch, A. Postula, A. Hemani “Grammar-based design of embedded systems“, Journal of Systems Architecture, Elsevier, vol. 47, n° 3-4, p. 225-40, avril 2001.

[Öbe99] J. Öberg, “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, Kista, Sweden, 1999.

[OCP] OCP-IP, http://www.ocpip.org.

[OCP03] Open Core Protocol International Partnership, “Open Core Protocol Specification 2.0”, 2003.

[OMI] Open Microprocessor systems Initiative, “OMI 324: PI-Bus”, draft standard, rev. 0.3d, 7 décembre 1996.

[O'Ni98] M. O'Nils, J. Öberg and A. J Jantsch “Grammar Based Modelling and synthesis of device drivers and bus interfaces”, in Proc. of the EuroMicro Workshop on Digital System Design, 1998.

[O'Ni01] M. O'Nils, A. J Jantsch “Device Driver and DMA Controller Synthesis from HW/SW Communication Protocol Specifications”, in Design Automation for Embedded Systems, vol. 6, n° 2, Kluwer Academic Publisher, 2001.

[Ope02] OpenCores Organization et Silicore Corporation, “WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores”, revision B.3, septembre 2002. Disponible sur http://www.opencores.org.

[OSCI] Open SystemC Initiative (OSCI), “SystemC Version 2.0 User's Guide. Update for SystemC 2.0.1”, 2002. Disponible sur : www.systemc.org.

[Pas98] R. Passerone, J. A. Rowson, “Automatic Synthesis of Interfaces between Incompatible Protocols”, in Proc. of DAC, 1998.

[Pav04] Yanick Paviot, “Partitionnement des services de communication en vue de la génération automatique des interfaces logicielles/matérielles”, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), préparée au laboratoire TIMA, juillet 2004.

[Pet03] Larry L. Peterson et B. S. Davie, “Computer Networks: A Systems Approach”, chapitre 1, 3eme edition, mai 2003, Morgan Kaufmann.

[Pet05] I. Petkov, P. Amblard, M. Hristov, A. A. Jerraya, “Systematic Design Flow For Fast Hardware/Software Prototype Generation From Bus Functional Model For MPSoC”, RSP 2005, Montréal, Canada, 8-10 June 2005.

[Phi01] Philips Semiconducteur, “Home Entertainment Engine – Nexperia pnx8525”. Disponible sur: http://www.philips.semiconductors.com.

[Python] Python, http://www.python.org.

[Rad03] A. Radulescu and K. Goossens, “Communication Services for Networks on Chip”, in Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, Shuvra Bhattacharyya and Ed Deprettere and Juergen Teich, editors. Marcel Dekker, 2003.

[Rad04] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema et P. Wielage, “An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming”, Proceedings of Design Automation and Test Conference in Europe, février 2004.

[Ros04] A. Rose, S. Swan, J. Pierce et J.-M. Fernandez, “Transactions Level Modeling in SystemC”, white paper, Cadence Design Systems Inc., 2004.

[Row97] J. A. Rowson, A. S.-Vincentelli, “Interface-Based Design”, DAC’97, USA.

[Sar04] A. Sarmento, W. Cesario, A. A. Jerraya, “Automatic Building of Executable Models from Abstract SoC Architectures”, 15th IEEE International Workshop on Rapid System Prototyping (RSP 2004), Genève, Suisse, juin 2004.

[Sar05] Adriano Sarmento, Lobna Kriaa, Arnaud Grasset, Mohamed-Wassim Youssef, Aimen Bouchhima, Frederic Rousseau, Wander Cesario, Ahmed Amine Jerraya, “Service Dependency Graph, an Efficient Model for Hardware/Software Interfaces Modeling and Generation for SoC Design”, CODES-ISSS 2005, New York, Etats-Unis, septembre 2005.

[Sea94] A. Seawright et F. Brewer, “Clairvoyant: A Synthesis System for Production-Based Specification”, IEEE Transactions on VLSI Systems, vol. 2, p. 172-185, juin 1994.

[Sea96] A. Seawright, U. Holtmann, W. Meyer, B. Pangrle, R. Verbrugghe, J. Buck “A System for Compiling and Debugging Structured Data Processing Controllers”, in Proc. of Euro-DAC, septembre 1996.

[Sea98] A. Seawright et W. Meyer, “Partitioning and optimizing controllers synthesized from hierarchical high-level descriptions”, proceedings of the 35th annual conference on Design automation, San Francisco, California, United States, p. 770 – 775, 1998.

[Sgr01] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sangiovanni-Vincentelli. “Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design”, Proceedings of the 38th Design Automation Conference, juin 2001.

[Shi02] D. Shin et D. Gajski, “Interface Synthesis from Protocol Specification”, rapport technique, avril 2002.

[Sie02] R. Siegmund et D. Müller “Automatic Synthesis of Communication Controller Hardware from Protocol Specifications”, IEEE Design & Test of Computers, p. 84-95, 2002.

[Sonics] SonicsStudio, http://www.sonicsinc.com/sonics/products/sonicsstudio.

[Spi04] SPIRIT Schema Working Group Membership, “SPIRIT-User Guide v1.0”, décembre 2004.

[STM04a] STMicroelectronics, “Nomadik – Open multimedia platform for net generation mobile devices”, technical article TA305, 2004. Disponible sur : www.st.com.

[STM04b] STMicroelectronics, “Nomadik ™ processor platform”, product flyer, 2004. Disponible sur : www.st.com.

[Syn01a] Synopsys, Inc., “Protocol Compiler ™ User Guide”, version 2001.08, août 2001.

[Syn01b] Synopsys, Inc., “Design Compiler ™ User Guide”, version 2001.08, août 2001.

[Tan96] A. S. Tanenbaum, “Computer Networks”, Prentice Hall, 1996.

[TEN] Tensilica, http://www.tensilica.com.

[UML] Rational Software Corp., “OMG Unified Modeling Language Specification, Version 1.3”, juin 1997. Disponible sur : http://www.rational.com/uml/resources/documentation/index.jsp

[Van02] G. Van Rossum, “Extending and Embedding the Python Interpreter”, F. L. Drake, Jr., editor, Release 2.2.2, 14 octobre 2002. Disponible sur : http://www.python.org.

[Ver96] S. Vercauteren, B. Lin and H. De Man “Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications”, in Proc. of DAC, juin 1996.

[Via98] C. Vial, B. Rouzeyre, “Cosynthèse matériel/logiciel : modélisation et synthèse des circuits d'interface”, dans “CODESIGN : Conception Conjointe Logiciel-Matériel”, édition Eyrolles, p. 125-146, 1998.

[VSIA] VSIA, http://www.vsia.com.

[Wie02] P. Wielage et K. Goossens, “Networks on Silicon: Blessing or Nightmare?”, Euromicro Symposium On Digital System Design (DSD 2002), Dortmund, Allemagne, septembre 2002.

[Win01] Drew Wingard, “MicroNetwork-Based Integration for SOCs”, DAC 2001.

[XML] W3C, “XML 1.0 Specification”, 2d ed., W3C Recommendation, octobre 2000. Disponible sur : http://www.w3c.org/XML.

[Zit02] A. Zitouni, M. Abid, K. Torki, R. Tourki, “Communication synthesis techniques for multiprocessor systems”, in International Journal of Electronics, vol.89, n°1, p. 55-76, janvier 2002.

[Zit93] M. Zitterbart, B. Stiller, A. N. Tantawy, “A Model for Flexible High-Performance Communication Subsystems”, IEEE Journal on Selected Areas in Communications, vol. 11, n°4, mai 1993.

Publications

Arnaud Grasset, Frédéric Rousseau, Ahmed Amine Jerraya, "Automatic Generation of Component Wrappers by Composition of Hardware Library Elements Starting from Communication Service Specification", 16th IEEE International Workshop on Rapid System Prototyping (RSP 2005), Montréal, Canada, Juin 2005.

Arnaud Grasset, Frédéric Rousseau, Ahmed Amine Jerraya, “Vers l’Automatisation de la Conception des Coprocesseurs de Communication pour les Systèmes Monopuces”, Journées nationales du réseau doctoral en Microélectronique (JNRDM 05), Paris, France, Mai 2005.

Adriano Sarmento, Lobna Kriaa, Arnaud Grasset, Mohamed-Wassim Youssef, Aimen Bouchhima, Frederic Rousseau, Wander Cesario, Ahmed Amine Jerraya, "Service Dependency Graph, an Efficient Model for Hardware/Software Interfaces Modeling and Generation for SoC Design", CODES-ISSS 2005, New York, Etats-Unis, Septembre 2005.

Arnaud Grasset, Frédéric Rousseau, Ahmed Amine Jerraya, "Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation", 15th IEEE International Workshop on Rapid System Prototyping (RSP 2004), Genève, Suisse, Juin 2004.

Arnaud Grasset, Frédéric Rousseau, Ahmed Amine Jerraya, "Génération des Interfaces de Communication pour Systèmes Multiprocesseurs Monopuces: de la Spécification des Services de Communication vers l'Implémentation RTL", Journées nationales du réseau doctoral en Microélectronique (JNRDM 04), Marseille, France, Mai 2004.

RÉSUMÉ

L'intégration dans une seule puce de un ou plusieurs processeurs et de composants matériels spécifiques permet le développement de systèmes complexes appelés systèmes monopuce. L'accroissement de la complexité de ces systèmes fait de la maîtrise de leurs conceptions un défi à relever par les concepteurs.

La réutilisation des composants dans ces systèmes est rendue difficile par leur hétérogénéité, notamment en terme de protocole et d'interface physique. Une solution est offerte par l'abstraction des communications entre les composants dans un modèle du système. Un flot de conception doit alors permettre de passer de cette représentation abstraite au circuit final dans lequel les composants du système sont connectés par des interfaces de communication à un réseau de communication.

Les contributions apportées par cette thèse à cette méthodologie sont la définition d'un modèle de spécification des interfaces de communication basé sur un graphe de dépendances de services, ainsi qu'une méthodologie pour la génération automatique d'interfaces de communication pour les systèmes monopuces. Cette méthodologie a amené au développement d'un outil de génération automatique de ces interfaces. L'approche proposée a été validée à travers deux expérimentations : une interface en charge de la détection d'erreurs de transmissions et une interface avec un bus AMBA pour la réalisation de primitives MPI.

MOTS-CLÉS

système monopuce, interface de communication, graphe de dépendances de services, génération automatique, modèle de communication, interface logiciel/matériel, découpage logiciel/matériel

TITLE

Network interface synthesis for system-on-chip: from specification to automatic generation

ABSTRACT

Integration in a single chip of one or more processors and specific hardware components allows the development of complex system, called systems-on-chip. With the increasing complexity of these systems, mastering of their designs is a challenge to take up by the designers. The re-use of the components in these systems is difficult due to their heterogeneity in terms of protocol and physical interface. A solution is the abstraction of the communications between the components in a system model. A design flow leads from this abstract representation to the final circuit where the components of the system are connected by network interfaces to a communication network. The contributions of this thesis are the definition of a specification model of the network interfaces based on a service dependency graph, as well as a methodology for the automatic generation of network interfaces for systems-on-chip. This methodology has driven to the development of an automatic generation tool of these interfaces. Two experiments allowed testing the approach with: an interface in charge of the error detection of transmissions and an interface with an AMBA bus for the realization of MPI primitives.

KEYWORDS

system-on-chip, network interface, service dependency graph, automatic generation, communication model, hardware/software interface, hardware/software partitioning

INTITULÉ ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN : 2-84813-081-4