CONCEPTION DES SYSTEMES MONOPUCE MULTIPROCESSEUR : DE LA SIMULATION VERS LA REALISATION

I. Petkov

UNIVERSITE JOSEPH FOURIER – GRENOBLE I

Sciences, Technologie & Médecine

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER

Spécialité : Microélectronique

présentée et soutenue publiquement

par

Ivan Doynov PETKOV

le 30 Janvier 2006

Titre :

Conception des systèmes monopuce multiprocesseur : de la simulation vers la réalisation

Directeur de thèse : Ahmed Amine Jerraya
Directeur de thèse : Marin Hristov Hristov
Co-directeur de thèse : Paul Amblard

JURY

Pr. Anne GUERIN-DUGUE	Président
Pr. Lionel TORRES	Rapporteur
Pr. Andrzej NAPIERALSKI	Rapporteur
Pr. Ahmed JERRAYA	Directeur de thèse
Pr. Marin HRISTOV	Directeur de thèse
Dr. Paul AMBLARD	Codirecteur de thèse

Thèse préparée en co-tutelle au sein du laboratoire TIMA – Grenoble, France et du laboratoire ECAD - Sofia, Bulgarie

Remerciements

Je tiens tout d'abord à exprimer mes sincères remerciements à Monsieur Ahmed Jerraya, Directeur de Recherché au CNRS, pour son accueil au sein de son équipe et pour avoir encadré cette thèse. Je le remercie pour ses conseils et pour toutes les opportunités qui m'ont été permises d'évoluer et d'apprendre un métier durant ces trois années de thèse.

De même, j'exprime ma plus grande gratitude à monsieur Paul Amblard, Maître de Conférences à l'Université Joseph Fourier de Grenoble, pour son encadrement, sa patience, sa disponibilité et sa bonne humeur. Je le remercie également pour l'aide qu'il m'a apporté durant ces trois ans de travail et plus spécialement pour finir cette thèse.

Je tiens tous particulièrement de remercier à Monsieur Marin Hristov, pour tout ce qu'il m'a appris, pour ça confiance en moi, pour m'avoir proposer de faire cette thèse et pour son soutien. Qu'il trouve ici, l'expression de ma profonde reconnaissance.

Je tiens à remercier à Madame Anne Guérin Dugué, Professeur à l'Université Joseph Fourier de Grenoble, pour m'avoir faire l'honneur de présider le jury. Je remercie également à Monsieur Lionel Torres, Professeur à l'Université de Montpellier et à Monsieur Andrzej Napieralski, Professeur à l'Université de Lodz, de m'honorer de leur participation au jury en tant que rapporteurs et pour les conseils apportés à l'amélioration de ce travail.

Je suis aussi reconnaissant aux thésards de l'équipe SLS, avec lesquels j'ai eu le plaisir de travailler : Wassim, Lobna, Aimen, Frédéric, Youngchul, Sand-Il Han, Lisane, Iuliana, Marius, Kati, Ferid, Patrice, Alexandre, Maxim, Lorenzo et Sao Hen. Ainsi que à mes collègues en Bulgarie qui ne m'ont jamais oublié : Rosen, Olga, Didi et Joro.

J'aimerais aussi remercier ceux qui ont partage mes repas, les pauses café ainsi que les moments difficiles durant ma thèse. Donc un grand merci à Arnaud, Arif, Marcio, Benaoumeur et Patricia.

Je remercie particulièrement de tous les amis sur Grenoble pour les soirées qui ont agrémenté ces dernières années, merci à Benj, Alex, Kiki, Frank, Herve, Xav, tous les Guillaume et bien sur à tous les bulgares : Natalia, Vania, Adi, Didi, Agata, Max, Delcho, et Boriana.

J'aimerais plus particulièrement remercie celle qui m'a le plus soutenue durant cette thèse et ces dernières huit ans. Celle qui m'a donne la force de continuer et d'espérer. Je remercie de tout mon cœur à Mariya pour l'amour qu'elle me donne.

Enfin je remercie énormément à ma famille, mon père, ma mère et ma sœur, qui m'a encouragée et soutenue durant ces années de thèse.

Table des matières

Liste des figures

Chapitre 1

CONCEPTION DES SYSTEMES MONOPUCE MULTIPROCESSEUR

Ce chapitre résume la problématique dans les méthodologies pour la conception des systèmes monopuce multiprocesseur. Il précise le contexte de ces travaux de thèse en se focalisant sur les systèmes sur puce, les différentes étapes du flot de conception et les niveaux d'abstraction utilisés dans la conception des systèmes embarqués multiprocesseur. Les objectifs à atteindre seront présentés dans ce chapitre, ainsi que les contributions apportées par cette thèse.

1. Introduction

Le progrès récent des technologies en microélectronique a permis l'intégration de plusieurs fonctionnalités sur un même substrat. Il est maintenant possible de créer des systèmes embarqués plus complexes, contenant plusieurs microprocesseurs, mémoires, réseaux de communication et circuits périphériques dans une seule puce. L'intégration de toutes ces fonctionnalités dans un circuit a augmenté la performance et réduit les coûts et l'énergie consommée mais elle a également introduit de nouveaux défis et des difficultés pour les ingénieurs des systèmes embarqués monopuce. Les systèmes embarqués modernes sont devenus complexes, impliquant des composants de nature hétérogènes tels que des composants logiciels : des applications embarquées, des systèmes d'exploitation, des interfaces matériel/logiciel, y compris des composants matériels comme des machines programmables, des microprocesseurs, des circuits intégrés spécifiques etc.

Pour traiter cette hétérogénéité, les ingénieurs de conception ont recherché une accélération du processus de conception de systèmes, réalisés avec des composants matériel/logiciel. Deux directions principales ont été étudiées : la conception au niveau système [JER05] et la conception basée sur des plateformes reconfigurables [VIN02]. Les résultats de ces études sont des nouvelles méthodologies de conception comme le raffinement automatique des interfaces matériel/logiciel [CES02], l'exploration d'architecture à partir de la spécification de système [BAG01], et la validation des systèmes par simulation conjointe du matériel et du logiciel [NAC99].

Ces nouvelles méthodes s'appuient sur des techniques et des outils largement utilisés en conception de circuits numériques, et proposent aux concepteurs de systèmes de se focaliser sur des choix d'architecture et de technologie. Toutes ces approches de conception utilisent des modèles d'abstraction de haut niveau des systèmes embarqués. Ces modèles développés à un niveau d'abstraction plus élevé sont employés pour accélérer la validation de la fonctionnalité et l'évaluation de la performance de divers composants de logiciel et de matériel constituant le système ciblé. Les étapes finales de conception sont alors faites par des outils automatiques.

Le présent travail cible les méthodes et les techniques de conception de systèmes et circuits numériques, depuis la spécification du système jusqu'à l'obtention des masques permettant de réaliser physiquement le circuit. Dans ce travail nous nous sommes concentré sur le passage d'un modèle de système de niveau de bus fonctionnel au niveau physique.

Ce chapitre a pour objet d'introduire les systèmes sur puce multiprocesseur et de donner un aperçu de la problématique de la conception des systèmes sur puce multiprocesseur allant de l'idée au produit.

2. Contexte : Conception des systèmes monopuce multiprocesseur

2.1. Les systèmes monopuce multiprocesseur

Les systèmes sur puce multiprocesseur ou MPSoC (Multiprocessor System on Chip), sont apparus grâce à l'évolution de la technologie de fabrication des circuits intégrés qui a permis l'intégration de plusieurs composants sur une seule puce. Ces systèmes ont rapidement pris une place importante dans le domaine de la microélectronique car ils intègrent dans un même circuit plusieurs processeurs, des nombreux composants numériques spécialisés et hétérogènes (mémoires, périphériques, unités de calcul spécifiques), du logiciel et souvent des circuits mixtes pour fournir un système intégré complet.

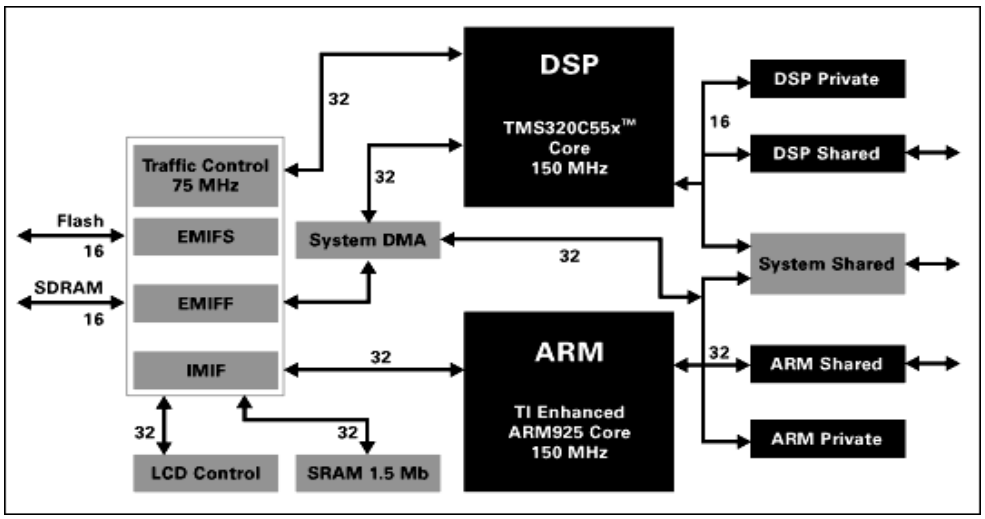


Figure 1.1 : L'architecture du système monopuce multiprocesseur OMAP5910 de la société Texas Instruments.

Les systèmes monopuce multiprocesseur sont généralement dédiés aux applications spécifiques dans des domaines différents comme par exemple dans le domaine de l'automobile, des télécommunications et du multimédia et leur diversité ne cesse pas d'accroître. La figure 1.1 montre l'architecture d'un tel système monopuce multiprocesseur. Le système monopuce multiprocesseur OMAP5910 [SWP01] développé par la société Texas Instruments est destiné principalement au marché de la téléphonie mobile mais peut aussi servir à la réalisation des applications multimédia pour des systèmes embraqués tel que les PDA, ou des systèmes de divertissement pour les automobiles.

2.2. Le concepts de base pour la modélisation des systèmes monopuce

La conception des systèmes sur puce est une tâche très difficile. Ils sont généralement spécifiques à une application ou à un domaine d'application et leur développement implique donc

une conception plus adaptée à l'application. Un système sur puce est donc construit selon des contraintes propres et il faut utiliser des composants spécifiques à certaines fonctionnalités.

L'abstraction est le concept de base le plus utilisé pour la modélisation des systèmes sur puce. Elle permet à la conception de circuits de commencer à partir d'un modèle abstrait fonctionnel. On dit aussi comportemental. Le fait de pouvoir décrire ce modèle de l'architecture globale du système et d'abstraire les composants, permet de réduire la complexité du système en cachant certaines caractéristiques des composants qui ne sont pas essentielles dans une ou l'autre des étapes de la conception. Ainsi on pourra simuler et valider le système à partir de ce modèle abstrait donnant la description structurelle du circuit et permettre la génération automatique de code du logiciel ou du matériel.

Il est possible de définir généralement cinq niveaux différents d'abstraction : le niveau système, le niveau transactionnel, le niveau de bus fonctionnel, le niveau transfert de registres et le niveau physique pour concevoir le matériel dans les systèmes sur puce. On peut définir des niveaux d'abstraction plus ou moins équivalents pour le logiciel. Au niveau système le circuit est représenté par un modèle abstrait qui implémente les tâches de l'application. Le deuxième niveau d'abstraction, le niveau transactionnel représente le système comme une architecture virtuelle composé par un ensemble des modules fonctionnels reliés par des canaux de communication abstraite, chaque module virtuel implémente une fonction logicielle ou matérielle. Au niveau de bus fonctionnel, toutes les fonctions matérielles de l'architecture sont associées aux ressources matérielles et les fonctions logicielles sont implémentées comme un programme utilisant un langage machine ou ISA (Instruction Set Architecture). Les systèmes de ce niveau possèdent des informations sur l'implémentation des protocoles de communication et des composants matériels utilisés dans les sous-systèmes matériels.

Le niveau transfert de registres ou RTL (Register Transfer Level) représente le système sous forme des registres et de circuits combinatoires, où des portes logiques élémentaires et bascules sont utilisés pour détailler le système. Enfin le dernier niveau englobe deux sous niveaux : le niveau circuit qui représente un schéma en transistors contenant aussi les caractéristiques physiques de chaque transistor (type, taille, paramètres) et le niveau le plus bas, le niveau physique qui représente le circuit sous forme de dessins de masques constitués par une liste de polygones des différentes couches physiques du circuit (diffusion, poly silicium, métal, etc.).

On reviendra plus en détail dans le chapitre 2 sur les différents niveaux d'abstraction utilisés dans la conception des systèmes monopuce.

2.3. Les différentes étapes de la conception

Chaque niveau d'abstraction, qui était introduit dans la section précédente, correspond à une phase de conception des systèmes monopuce. Le passage systématique d'un niveau d'abstraction à un autre détermine un flot de conception. Ce flot de conception est généralement divisé en deux parties principales : une partie frontale (front-end), qui consiste à raffiner une spécification initiale pour produire une description de haut niveau du système et une partie dorsale (back-end) qui réalise les étapes de conception physique du matériel pour produire la topologie du système.

La partie frontale de flot de conception s'intéresse à la fonctionnalité, indépendamment de l'implémentation finale. Elle sera utilisée pour explorer des solutions différentes afin de fixer une architecture finale qui sera réalisée. Ce qui permet de fixer les principales contraintes du produit à réaliser. La partie dorsale comprend la conception des parties matérielles en passant par la conception logique et la conception physique. La figure 1.2 montre les différentes étapes de conception des systèmes monopuce multiprocesseur définit par [SAS04].

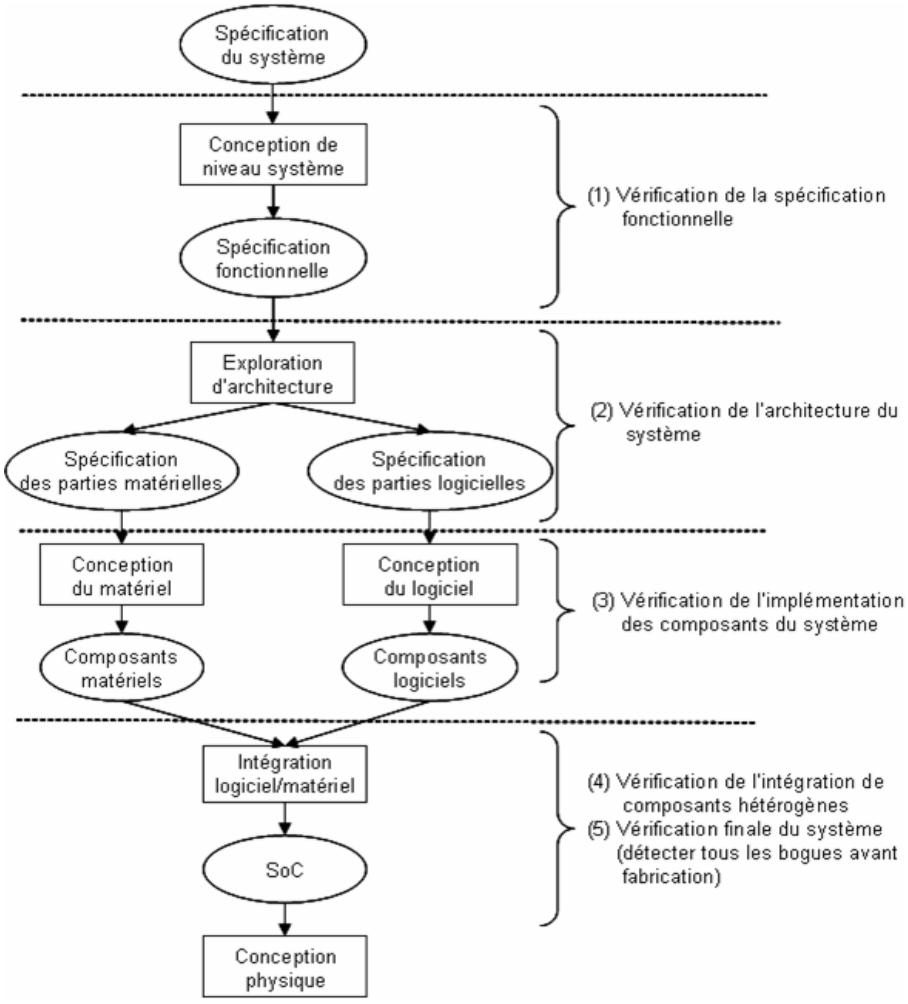


Figure 1.2 : Le flot de conception des systèmes monopuce.

Le point de départ de ce flot de conception est une spécification du système à concevoir. Cette spécification est généralement transformée pour obtenir une spécification comportementale exécutable. La première étape présente la phase de conception au niveau système. A ce niveau de conception, le circuit est décrit par un ensemble de tâches communiquant par des services de communication. Il est utilisé pour valider par simulation la fonctionnalité et les relations temporelles entre les composants.

La deuxième étape consiste à allouer les tâches dans des ressources matérielles ou logicielles capable de réaliser les fonctions décrites. C'est l'espace d'exploration d'architectures qui détermine l'implémentation logicielle ou matérielle de tous les composants. Il faut donc déterminer les parties du système qui seront implémentées en logiciel (programme s'exécutant sur un processeur) et celles qui le seront en matériel (composants physiques). On obtient alors un ensemble de spécifications pour les parties logicielles, matérielles et les interfaces entre les différents composants de l'architecture.

Cette macro-architecture peut être utilisée par les différentes équipes pour réaliser les composants matériels, les composants logiciels et l'intégration des différents composants. Le résultat du raffinement de ces spécifications est une microarchitecture du système décrit au niveau de bus fonctionnel. Ce modèle contient tous les détails de la communication entre les composants. Les couches de communication logicielle peuvent comporter un système d'exploitation spécifique. Les couches de communication matérielles comportent les bus et autres dispositifs permettant de réaliser le transfert des informations entre les composants. Les blocs matériels sont raffinés au niveau cycle près. Pour établir la communication entre les composants matériels, des interfaces de communication sont nécessaires pour adapter les différents protocoles et le type de données. Entre un processeur et le réseau de communication, ces adaptateurs de communication font la connexion entre les composants matériels et le réseau de communication. Ils peuvent être complexes incluant une partie logicielle (pilotes des contrôleurs d'entrées/sorties) et une partie matérielle (composants d'interface). Les composants logiciels communiquent entre eux et avec l'extérieur via des appels système au système d'exploitation ou OS (Operating System).

La phase d'intégration logiciel/matériel est le passage du niveau de bus fonctionnel au niveau transfert de registres. Cette étape de conception utilise la synthèse comportementale ou l'assemblage automatique des composants existants pour permettre de gagner du temps par rapport à une conception complète de tous les composants matériels du système. Elle va générer une représentation structurelle du système constitué d'une partie opérative et une partie de contrôle pour le transfert et le calcul des données (pour le matériel) et un programme en assembleur pour la partie logicielle.

Cette représentation structurelle est employée d'une part pour vérifier que le logiciel s'exécute correctement sur les composants programmables ou sur les processeurs ciblés, mais aussi pour valider l'intégration correcte et les échanges d'informations exactes entre les composants matériels.

La dernière phase est l'étape de conception physique. La synthèse au niveau transfert de registres permet de générer la vue structurelle du circuit. On s'appuie alors sur une bibliothèque de portes logiques existante (bibliothèque d'un fondeur ou cellules de FPGA) pour trouver la correspondance avec les équations booléennes (conversion technologique). La synthèse de niveau circuit consiste à transformer un ensemble d'équations en un schéma en transistors.

3. Problématique : La complexité et les difficultés de la conception des systèmes sur puce

La complexité des systèmes monopuce est devenue telle qu'il est impossible de continuer à les concevoir au niveau RTL, où il faut préciser chaque détail du comportement des composants. Le grand défi en ce moment pour les ingénieurs est de réussir à maîtriser la complexité lors de la conception de ces systèmes et d'arriver à une conception rapide des systèmes monopuce sous de fortes contraintes de qualité et de temps de développement. Pour dépasser ce défi, les nouvelles méthodes de conception sont basées sur des concepts d'abstraction de haut niveau.

La problématique de cette thèse est de comprendre les difficultés de la conception des systèmes sur puce commençant à un niveau d'abstraction élevé et d'essayer de trouver des méthodes ou techniques pour faciliter et accélérer leur développement. ***Nous nous sommes posé comme objectif d'étudier différentes méthodologies de prototypage des systèmes monopuce et les problèmes liés avec les niveaux d'abstraction et les outils de conception qui génèrent des différences dans le comportement des composants à chaque niveau d'abstraction.*** La contribution attendue est de proposer une méthode d'intégration efficace permettant de lier la conception au niveau système au niveau RTL. Cette méthode permettra de compléter le flot et les outils de conception de niveau système de l'environnement ROSES [CES02] développés dans le groupe SLS au laboratoire TIMA.

Pour mieux comprendre cette problématique, une expérimentation a été conduite : la conception RTL et physique d'un système monopuce multiprocesseur d'encodage vidéo. Cette expérimentation fait suite à des expériences plus simples dont certaines faites au laboratoire ***ECAD de l'Université Technique de Sofia***.

3.1. Description des composants aux différents niveaux d'abstraction

Une première difficulté de conception réside dans les différents modèles utilisés pour décrire les composants à chaque niveau d'abstraction. La conception de systèmes monopuce multiprocesseur consiste à raffiner, optimiser et synthétiser les spécifications du circuit en descendant dans les niveaux d'abstraction. L'étape de raffinement d'un niveau d'abstraction au suivant consiste à donner plus de détails sur la réalisation des composants du système. On décompose ainsi le problème de conception en un ensemble de petits problèmes traités consécutivement, en donnant la possibilité à l'ingénieur de se focaliser sur différentes difficultés séparément jusqu'à la réalisation physique du circuit. Le chemin de raffinement dans la conception n'est pas unique et il dépend des connaissances du concepteur, des outils et des composants disponibles, du type d'architecture ciblée, ainsi que de la méthode de conception choisie, et il peut différer pour chaque composant du système.

Les techniques de raffinement consistent en la génération d'un modèle structurel à partir d'un modèle comportemental. La synthèse est triviale quand il s'agit de générer un modèle structurel, sans se préoccuper de la performance. Mais elle s'avère beaucoup plus fastidieuse quand il s'agit d'optimiser et de générer un circuit plus performant. L'optimisation conduit généralement à l'introduction de la logique qui n'est pas explicitement présentée dans la description comportementale pour produire la description structurelle (logique supplémentaire, éléments de bibliothèque spécifique, etc.). Alors une modification de la description initiale est nécessaire. De plus, l'étape de synthèse regroupe plusieurs étapes de raffinement, synthèse, analyse, et optimisation, ce qui rend difficile la compréhension du flot de conception.

3.2. Les techniques d'intégration des composants matériel/logiciel aux différents niveaux d'abstraction

On peut définir un ensemble d'étapes pour la conception des systèmes monopuce, qui correspond à un flot de conception. Le flot de conception réside dans la transformation d'un description du système à un niveau d'abstraction plus élevé à un descriptif à un niveau d'abstraction plus bas.

En effet, il est difficile de définir un flot de conception unique et universel. Car pour chaque étape de la conception sont associés plusieurs outils de conception qui utilisent plusieurs représentations différentes du système pour un niveau d'abstraction, ce qui signifie qu'il existe plusieurs façons de décrire le système pour un même niveau. Il est aussi possible de représenter un système comme un ensemble de composants, chacun décrit à un niveau d'abstraction différent. On peut ajouter aussi les nombreux langages qui sont développés pour couvrir les étapes de

conception. Parmi les plus utilisés sont VHDL et Verilog qui supportent les concepts dédiés à la description et à la simulation de matériel. On retrouve aussi des langages comme SystemC et SpecC dérivés des langages de programmation (C, C++), qui supportent les notions de temps, de concurrence et de communication, et qui sont essentiellement consacrés à la conception comportementale au niveau système. A tout cela s'ajoute les différents niveaux d'abstraction possibles pour le temps.

C'est pourquoi, les objectifs de cette thèse sont d'étudier et appliquer les différentes techniques d'intégration des composants matériel et logiciel. Le premier objectif est de proposer une méthodologie systématique de passage du niveau de bus fonctionnel au niveau transfert de registres. Le deuxième objectif est de compléter le flot de conception du groupe SLS et de faire apparaître les problèmes pendant l'étape de la conception physique dûs à la génération automatique de la description RTL.

4. Contribution de la thèse

Ces travaux de thèse présentent trois contributions. Elles trouvent place dans la conception des systèmes multiprocesseurs hétérogènes à l'étape d'intégration de matériel et de logiciel à partir d'un modèle abstrait et dans le prototypage des applications monopuce multiprocesseur.

4.1. Définition d'une méthode pour faciliter l'intégration de matériel et de logiciel à partir d'un modèle au niveau de bus fonctionnel

Pour réduire le temps et l'effort de développement du prototype, le premier apport de cette thèse contribue à la définition d'une nouvelle méthodologie systématique d'intégration des systèmes sur puce (SoC) partant d'un prototype virtuel d'une application pour obtenir un prototype du matériel synthétisable au niveau RTL. L'objectif de cette contribution est de faciliter le passage entre deux niveaux d'abstraction en utilisant une méthode systématique pour la transformation. Cette méthode part d'un modèle du système au niveau de bus fonctionnel et s'arrête à un modèle au niveau RTL. Le flot de conception d'intégration de SoC fournit un chemin systématique pour produire le code du logiciel considérant les détails de bas niveau, tels que la configuration et l'initialisation du processeur, ainsi que la configuration de la mémoire. L'architecture synthétisable est obtenue en remplaçant les composants fonctionnels par leurs équivalents en RTL en réutilisant une bibliothèque de composants existante.

4.2. Formalisation de l'implémentation et de la réalisation physique des composants matériel pour des systèmes sur puce multiprocesseurs

L'objectif initial de cette contribution était d'examiner le chemin et les liens entre le flot de conception physique et le flot de conception de haut niveau. Dans le processus de travail, il s'est concrétisé sur l'étude des modèles RTL générés automatiquement et la proposition d'un modèle d'intégration physique efficace. La motivation des travaux effectués est basée autour des problèmes liés aux outils de synthèse de niveau système qui génèrent automatiquement des descriptions inefficaces au niveau RTL. Cela est dû aux différences dans les représentations des composants matériels aux différents niveaux d'abstraction.

La deuxième contribution traite ce problème en formalisant la réalisation et la synthèse des composants matériels en définissant certaines techniques de codage. La contribution cible un guide pour éviter les problèmes qui peuvent apparaître pendant l'étape de génération automatique de la description RTL.

La contribution de ce travail trouve place dans le domaine de la modélisation des systèmes multiprocesseurs hétérogènes. Il s'agit plus précisément de l'étape de réalisation physique des systèmes générés automatiquement comme les interfaces de communication d'une application VDSL.

4.3. Expérimentation avec un encodeur MPEG4

Une dernière contribution de cette thèse est l'application des deux premières contributions par la conception d'une application de codage de vidéo en temps réel. Le groupe SLS du laboratoire TIMA a développé un encodeur de vidéo en temps réel à partir de l'implémentation OpenDivX [MAY00] du standard de codage MPEG4 [KOE2]. L'architecture de cette application est le résultat de plusieurs travaux de recherche dirigés dans le groupe SLS. Elle est composée de trois sous-systèmes d'entrée nommée DivX implémentant la transformation DCT (Discrete Cosinus Transformation) et la quantification, l'évaluation et la compensation de mouvement, et un sous-système nommée VLC (Variable Length Coder) réalisant le codeur d'entropie. Cette architecture a servi notamment à expérimenter les méthodes et les outils de l'environnement ROSES du groupe.

Une grande partie de mes travaux de thèse était consacrée aux détails d'intégration de cette architecture au niveau RTL à partir de l'architecture au niveau système, et à la conception physique de chacun des quatre sous-systèmes, qui étaient implémentés autour du processeur ARM946E-S.

5. Plan de la thèse

La suite de ce mémoire est organisée en cinq chapitres.

Chapitre 1 : Ce chapitre a présenté la problématique dans les méthodologies de conception, ainsi que les différentes étapes de flot de conception et les niveaux d'abstraction utilisés dans la conception des systèmes embarqués multiprocesseur.

Chapitre 2 : Le chapitre 2 fait un état de l'art sur les composants matériels, les composants logiciels et sur quelques architectures commerciales utilisés pour des systèmes monopuce multiprocesseur. On discutera aussi en détail les différents niveaux d'abstraction et certains des outils de conception.

Chapitre3 : Ce chapitre présentera en détail une des contributions de ces travaux de thèse. Il décrit une méthode d'intégration de matériel et de logiciel commençant par un modèle d'abstraction d'un système sur puce au niveau de bus fonctionnel vers le niveau RTL.

Chapitre 4 : Dans ce chapitre, il sera présenté les étapes de la conception pour le passage du niveau RTL à la description physique d'un système monopuce.

Chapitre 5 : Les détails sur la vérification et les différentes étapes de prototypage physique de l'application de codage de vidéo seront discutés dans le chapitre 5.

Chapitre 6 : Le dernier chapitre est consacré à l'analyse et aux perspectives de ce travail.

Chapitre 2

ARCHITECTURES ET METHODES POUR LA CONCEPTION DES SYSTEMES MONOPUCE MULTIPROCESSEUR

Ce chapitre introduit les méthodologies pour la conception des systèmes monopuce multiprocesseur. Il commencera par une présentation des composants constituant les architectures des systèmes monopuce multiprocesseurs. Le chapitre va ensuite présenter les différents niveaux d'abstraction, langages et outils utilisés pour leur conception.

I. Les architectures des systèmes monopuce multiprocesseur

La première section de ce chapitre abordera en détail les architectures des systèmes sur puce ou SoC. Les niveaux d'abstraction, les langages et les outils utilisés pour la conception de ces sous-systèmes seront ensuite présentés dans la deuxième section de ce chapitre. Dans cette première section, nous allons passer en revue les principes généraux des architectures monopuce multiprocesseur en définissant précisément la terminologie qui sera utilisée dans cette thèse. On va aussi analyser leurs structures et leurs fonctionnalités principales, ainsi que les composants de base constituant ces architectures.

1. Introduction

Dans le présent ouvrage le terme *architecture* est utilisée pour caractériser la structure fondamentale d'un système électronique. La *structure* est une représentation de l'interconnexion des modules composant un système embarqué. Elle est définie par les caractéristiques et le comportement des composants, des interfaces et des liens d'interconnexion qui la constituent.

Un *module* du système implémente une ressource logicielle ou matérielle comme par exemple un processeur ou une mémoire. Chaque module peut être, soit hiérarchique et donc composé par des modules plus simples, soit être représenté par un comportement élémentaire. Le comportement du module peut implémenter une tâche élémentaire ou un algorithme complexe.

Chaque module peut être associé aux différents *modèles*. Un *modèle* représente une vue particulière plus ou moins abstraite du module. Les modèles sont utilisés pour décrire le comportement et la communication des composants, pour pouvoir ensuite analyser et vérifier les propriétés des composants du système.

Par défaut, en parlant d'architecture matérielle on pense immédiatement à une architecture autour d'un seul processeur. Dans le présent ouvrage, le terme architecture est aussi utilisé pour représenter l'architecture générique d'une application multiprocesseur, qui est constitué d'un ensemble de plusieurs sous-systèmes basés autour d'un processeur.

Tous ces sous-systèmes sont interconnectés autour d'un réseau de communication qui sert à acheminer le flux d'informations entre eux. De son coté chaque sous-système peut être de type différent. Une telle architecture générique est montrée sur la figure 2.1.

Dans cette thèse, les architectures sont étudiées par rapport à une seule vue : la vue dite *ressource* décrivant l'implémentation physique du système.

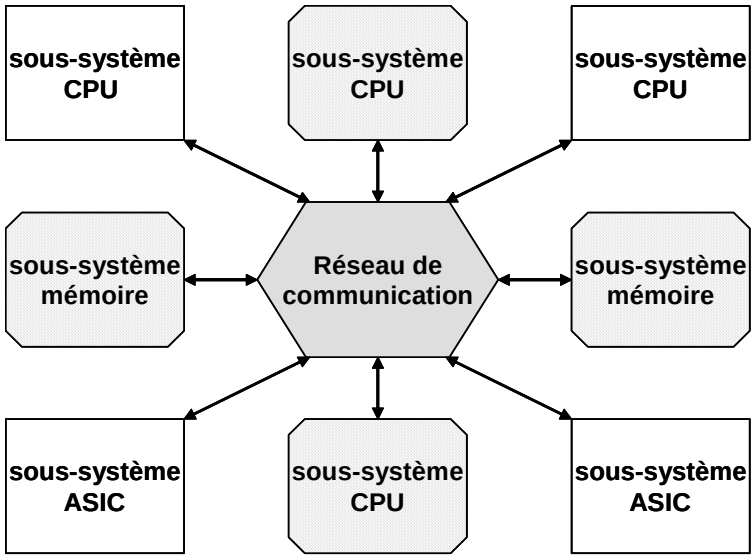


Figure 2.1 : L'architecture générique d'un système multiprocesseur.

La *vue ressource* se décrit par une architecture dite matérielle ou physique. Elle définit les composants matériels ou physiques utilisés comme ressource ainsi, elle représente l'infrastructure d'exécution du système. Toutes les ressources et liens d'interconnexion sont définis par leurs caractéristiques physiques, comportements, protocoles etc.

A chaque composant, il est possible d'assigner trois vues différentes :

- ✓ Une vue *comportementale* (behaviour), représente ce que le système fait sous la forme d'un comportement entrée/sortie (boîte noire). Toute décomposition hiérarchique est purement fonctionnelle et n'implique pas forcément une structure.
- ✓ Une vue *structurelle* représente comment le système est logiquement construit sous la forme d'une interconnexion de composants. Cette vue ne prend pas en compte l'aspect géométrique.
- ✓ Une vue *physique* (ou géométrique) représente comment le système est réellement construit. Elle prend en compte les aspects de taille, de forme et de position des composants.

La suite de cette section va présenter les sous-systèmes des applications multiprocesseur par rapport aux composants de base constituant leurs architectures. Un sous-système générique possède trois éléments principaux : *matériel, logiciel et une architecture*.

On commencera par les composants de matériel en définissant les fonctions de base qu'on retrouve dans ces sous-systèmes et ensuite on va distinguer les quatre familles de composants qui les implémentent. On précisera ceux qui sont utilisés dans la partie expérimentale de la thèse. Le deuxième paragraphe abordera les composants logiciels de base. Ce paragraphe ne prétend pas couvrir l'ensemble des composants logiciels existants. Quelques architectures monopuce

multiprocesseur commerciales seront étudiées dans le troisième paragraphe afin de définir un modèle d'architecture régulière par ces travaux.

2. Définition des fonctions de base dans les sous-systèmes

Les fonctions logiques issues des mathématiques et plus précisément de l'algèbre booléenne sont les éléments de base pour la conception de l'électronique numérique. Elles sont utilisées en électronique numérique sous forme d'automates et de bascules. Ces automates et bascules sont construits à partir de plusieurs portes logiques reliées entre elles.

Si on essaie de remonter un peu dans l'abstraction des systèmes numériques on pourrait définir les fonctions système issues des éléments des sous-systèmes monopuce multiprocesseur. Alors si on examine un modèle universel (voir la figure 2.2), qui sert d'une architecture de base à la plupart des sous-systèmes multiprocesseur actuels, on va remarquer que l'architecture d'un tel sous-système monopuce est un assemblage des composants suivants : des composants de calcul, des composants de mémorisation et des composants de communication.

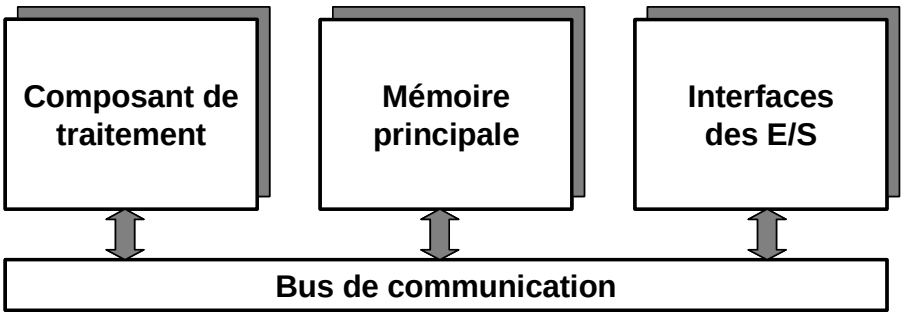


Figure 2.2 : L'architecture de base d'un système embarqué.

Alors on peut conclure que le comportement de chaque composant qui constituent cette architecture peut être caractérisé par une des quatre fonctions de base : *la fonction de traitement des données, la fonction de mémorisation des données, la fonction d'interfaçage des entrées/sorties et la fonction de transfert des informations*.

2.1. Fonction de traitement des données

La fonction de calcul ou de traitement des données permet l'implémentation plus ou moins flexible d'un algorithme de contrôle et ou de traitement de données. Elle est chargée d'interpréter et d'exécuter les instructions d'un programme ou d'un algorithme, de lire ou sauvegarder les résultats dans la mémoire et de communiquer avec d'autres composants. Aujourd'hui on peut définir deux types de composants pour effectuer les traitements numériques : **composants programmables ou modifiables** et **composants câblés ou non modifiables**.

2.2. Fonction de mémorisation

La fonction de mémorisation décrit le comportement des composants de stockage des données qui permettent de mémoriser pour une durée variable des données à traiter, des résultats temporaires, ainsi que des informations de configuration telles que des instructions de processeur.

Selon le type d'opération qu'on peut effectuer sur la mémorisation, lecture ou écriture d'une donnée on peut distinguer deux types de mémoires. Les mémoires qui n'autorisent que des accès en lecture et les mémoires qui permettent les accès en lecture et en écriture.

2.3. Fonction d'interfaces d'entrées/sorties

Elle permet d'assurer la communication entre le composant de calcul et les composants périphériques d'un système. Chaque composant périphérique sera relié au système par l'intermédiaire d'une interface dont le rôle est de :

- ✓ Connecter les composants périphériques au bus de données,
- ✓ Gérer les échanges entre le composant de traitement et les composants périphériques.

2.4. Fonction de communication

Les liens d'interconnexions qui réalisent les liaisons de transfert des données entre les composants de calcul et de mémorisation, se caractérisent aussi par une fonction, la fonction de communication. Cette fonction implémente des ressources de routage et de transport de l'information, et aussi la fonction arbitrage. On retrouve trois types de communications en parallèle dans un *sous-système* :

- ✓ le bus de données : une communication bidirectionnelle qui assure le transfert des informations entre le composant de calcul et son environnement,
- ✓ le bus d'adresses : un transfert unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire,
- ✓ le bus de commande : la communication qui assure la synchronisation des flux d'informations sur les bus des données et des adresses.

3. Les composant matériels de base constituant les architectures des systèmes sur puce multiprocesseur

3.1. Les composants de calculs

Le choix de l'un des deux types de composants de calcul, *composants programmables ou composants câblés*, pour un système va dépendre de l'ensemble des critères fixés lors de la

spécification. La solution d'un composant dédié non modifiable permet d'atteindre les meilleurs temps de traitement pour un minimum de surface et de puissance consommée. De plus les coûts initiaux relativement importants de ces circuits peuvent être amortis si le nombre de pièces fabriquées est élevé. Cependant ces architectures étant dédiées, elles manquent de flexibilité. Pour cette raison les concepteurs choissent souvent se tourner vers des solutions plus souples comme des composants programmables ou des processeurs.

Un processeur est un circuit intégré complexe dédié pour l'interprétation et l'exécution des instructions d'un programme. Il est chargé d'organiser les tâches du programme et d'assurer leur exécution. Il doit aussi prendre en compte des informations extérieures au système et assurer leur traitement. C'est le cœur du système.

Les architectures basées sur ce composant de calcul utilisent un programme qui définit les opérations à réaliser et les opérandes à aller chercher en mémoire. Elles n'utilisent en général qu'un faible nombre de ressources de calculs et de registres qui sont réutilisées dans le temps. Ces architectures sont généralement très flexibles, mais ne permettent pas toujours d'atteindre des performances élevées.

Dans cette thèse, tous les sous-systèmes matériels sont étudiés et conçus autour d'un processeur. C'est pour cela que l'on va détailler les caractéristiques principales des processeurs et que l'on va présenter certaines classes de processeur : généraux, spécifiques et personnalisables.

3.1.1. Les caractéristiques principales des processeurs

Jeu d'instructions : Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le processeur pourra exécuter. Il va donc en partie déterminer l'architecture du processeur à réaliser et notamment celle du séquenceur. Le processeur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. A un même jeu d'instructions peut correspondre un grand nombre d'implémentations différentes du processeur.

Cycle d'exécution d'une instruction : Le traitement d'une instruction peut être décomposé en trois phases :

- ✓ Recherche de l'instruction à traiter,
- ✓ Décodage de l'instruction et recherche de l'opérande ou des opérandes,
- ✓ Exécution de l'instruction et désignation de l'instruction suivante.

Performances d'un processeur : On peut caractériser la puissance d'un processeur par le nombre d'instructions qu'il est capable de traiter par seconde. Pour cela, on définit :

- ✓ le CPI (Cycle Par Instruction) qui représente le nombre moyen de cycles d'horloge nécessaire pour l'exécution d'une instruction pour un processeur donné,

- ✓ le MIPS (Millions d'Instructions Par Seconde) qui représente la puissance de traitement du processeur.

Pour augmenter les performances d'un processeur, on peut donc soit augmenter la fréquence d'horloge (limitation matérielle), soit diminuer le CPI (choix d'un jeu d'instruction adapté).

Temps d'exécution. Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer. Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage. Il est plus long d'accéder à la mémoire principale qu'à un registre du processeur. La durée d'un cycle dépend de la fréquence d'horloge du séquenceur.

Un ensemble d'améliorations de l'architecture des processeurs vise à diminuer le temps d'exécution du programme :

- ✓ Architecture pipeline : L'architecture pipeline permet d'améliorer l'efficacité du processeur. En effet, lorsque la première étape de l'exécution d'une instruction est achevée, l'instruction entre dans la seconde étape de son exécution et la première phase de l'exécution de l'instruction suivante débute. Il peut donc y avoir une instruction en cours d'exécution dans chacune des étapes et chacun des composants du processeur peut être utilisé à chaque cycle d'horloge. L'efficacité est maximale. Le temps d'exécution d'une instruction n'est pas réduit mais le débit d'exécution des instructions est considérablement augmenté. Un processeur pipeliné se caractérise par le nombre d'étapes utilisées pour l'exécution d'une instruction.
- ✓ Cache mémoire : Une des solutions utilisées pour diminuer la latence entre le temps de réponse de la mémoire et la vitesse de calcul du processeur est d'intégrer une mémoire très rapide entre le processeur et la mémoire. Elle est appelée mémoire cache. On compense ainsi la faible vitesse relative de la mémoire en permettant au processeur d'acquérir les données à sa vitesse propre. Cette mémoire cache est réalisée à base de mémoires SRAM de petite taille. Le principe des caches est très simple et leur présence est transparente pour le processeur, qui envoie toutes ses requêtes comme s'il s'agit d'une mémoire principale.
- ✓ Architecture superscalaire : Une autre façon de gagner en performance est d'exécuter plusieurs instructions en même temps. L'approche superscalaire consiste à doter le processeur de plusieurs unités de traitement travaillant en parallèle. Les instructions sont alors réparties entre les différentes unités d'exécution. Il faut donc pouvoir soutenir un flot important d'instructions et pour cela disposer d'un cache performant.

3.1.2. Processeur d'usage général

Le processeur ARM946-E-S de la société ARM. L'ARM946E-S [ARM03] est un processeur d'usage général. Il est ciblé pour une vaste gamme d'applications embarquées, où la haute performance, les bas coûts de système, la petite taille en surface et la basse consommation d'énergie sont d'une grande importance

L'ARM946E-S est un processeur d'architecture Harvard avec des mémoires caches. Il combine un coeur de processeur ARM9E-S avec un sous-système de mémoire configurable. Son architecture interne inclut :

- ✓ Un coeur d'unité centrale de traitement de la famille ARM9 avec un jeu d'instruction ARMv5TE, amélioré pour fonctionner avec des instructions ARM de 32 bits et des instructions Thumb de 16 bits et un multiplicateur spécialement amélioré pour l'exécution optimisée des instructions DSP (Digital Signal Processing). Le cœur ARM9 contient aussi une architecture spécifique supplémentaire pour le débogage en temps réel. Ceci permet le traitement des exceptions critiques tout en déboguant le système.
- ✓ Soutien des mémoires externes « collées » au processeur ou TCM (Tightly Coupled Memory), permettant l'accès à la mémoire dans un seul cycle. Une interface TCM est prévue pour chacun des blocs mémoires d'instructions et de données. Les tailles des blocs mémoires TCM d'instruction et de données sont spécifiquement implémentées et peuvent atteindre jusqu'à 1MB.
- ✓ Des caches d'instructions et de données. Le sous-système mémoire du processeur ARM946E-S peut être facilement modifié pour permettre plusieurs combinaisons de tailles de cache jusqu'à 1MB.
- ✓ Une unité de protection qui permet de protéger la mémoire d'une façon simple, idéale pour des applications embarquées de commande.
- ✓ Une interface de bus AHB. Les interfaces de processeur ARM946E-S avec le reste du système utilisent des bus d'adresse et de données unifiés compatibles avec le standard de bus AMBA.
- ✓ Soutien d'un coprocesseur externe ou un autre accélérateur matériel spécifique permettant d'ajouter des fonctions supplémentaire comme traitement à virgule flottante. Pour le soutien d'un coprocesseur, les bus d'instructions et de données sont exportés avec les signaux simples du standard de poignée de main (handshake).
- ✓ Soutien de la méthodologie BIST (Bilt-In-Self-Test) pour le TCM et les caches.
- ✓ Une interface spéciale pour soutenir le traçage en temps réel des instructions et des données en utilisant le module externe ETM (Embedded Trace Macrocell).

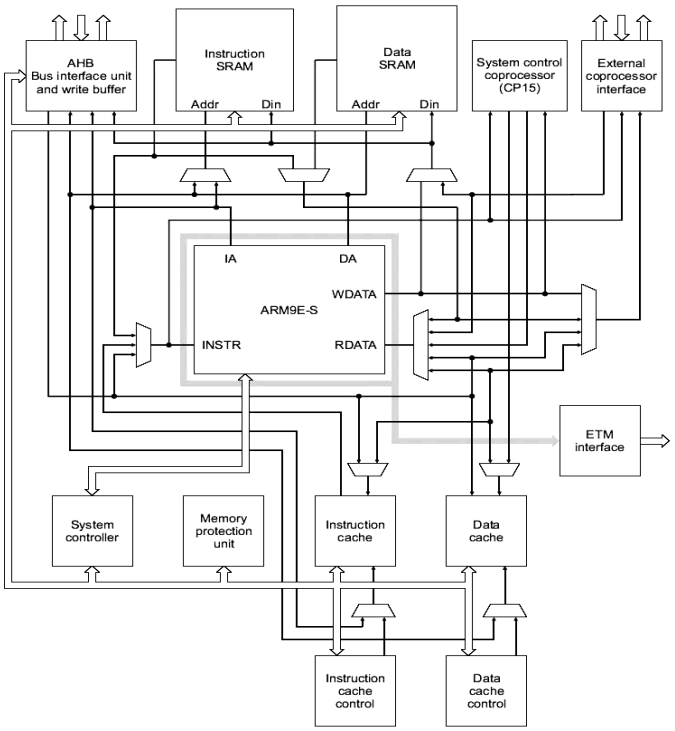


Figure 2.3 : L'architecture interne du processeur ARM946E-S.

Avec ce sous-système complet travaillant à la même fréquence que le processeur, les concepteurs des systèmes sur puce ont la possibilité de se concentrer sur les problèmes de conception spécifique à leur système. La nature synthétisable du processeur facilite son intégration dans des technologies ASIC. Nous utiliserons ce processeur dans la quatrième partie de cette thèse décrivant la partie expérimentale.

3.1.3. Processeur d'usage spécifique

Le processeur d'usage spécifique est beaucoup plus spécialisé. Alors qu'un processeur générique n'est pas conçu pour une application spécifique, le processeur de signal ou le processeur DSP (Digital Signal Processor) est optimisé pour effectuer du traitement numérique du signal (calcul de FFT, convolution, filtrage numérique, etc.).

Les domaines d'application des DSP étaient à l'origine les télécommunications et le secteur militaire. Aujourd'hui, les applications se sont diversifiées vers le multimédia (lecteur CD, MP3, etc.) l'électronique grand public (télévision numérique, téléphone portable, etc.), l'automatique, l'instrumentation, l'électronique automobile, etc.

Une limitation principale aux applications sans fil est la consommation d'énergie. C'est pourquoi on utilise des processeurs DSP pour exécuter une tâche plus efficacement, car si une tâche de traitement peut être réalisée plus efficacement, le processeur DSP peut rester longtemps dans l'état de puissance bas, ou mode IDLE. Cette intégration rapporte non seulement une diminution de la puissance mais aussi de la surface.

Le processeur DSP TMS320C54x de la société Texas Instrument : Actuellement, un grand nombre de téléphones portables possède à l'intérieur un processeur DSP de la famille TMS320C54x de Texas Instrument [BUD00]. La famille TMS320C54x des processeurs DSP fournit une combinaison entre une basse consommation d'énergie, une haute performance, et une rentabilité pour satisfaire les besoins d'une variété d'applications et des équipements de communication et de transmission.

Le processeur TMS320C54x est conçu pour exécuter jusqu'à 100 millions d'instructions par seconde (MIPS) tout en n'utilisant que 0.45 mA/MIPS. Cette performance efficace par rapport à la puissance consommée est exigée dans toutes les applications sans fil et dans toutes les applications à pile ou à batterie.

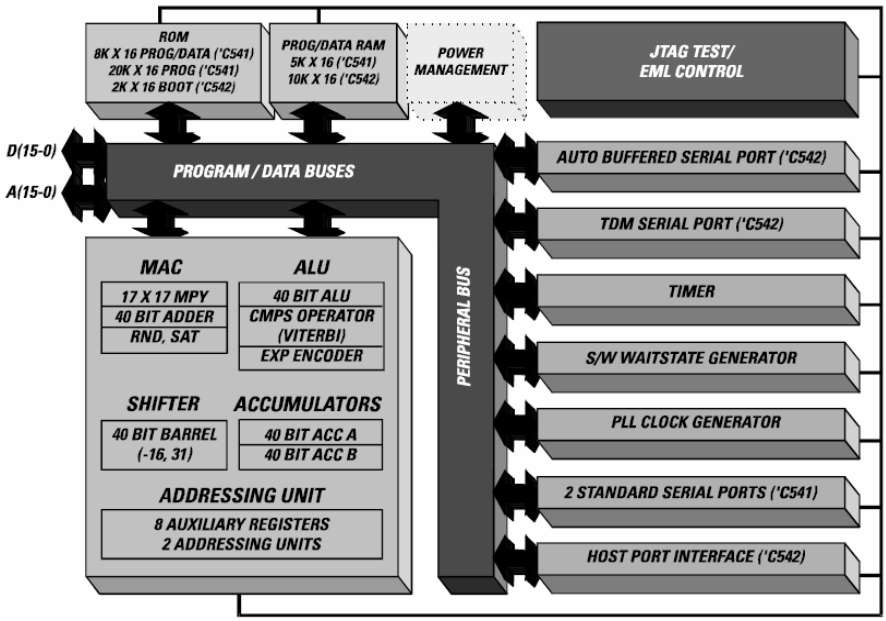


Figure 2.4 : L'architecture de processeur DSP TMS320C54x de la société TI.

La performance élevée du processeur TMS320C54x est devenue possible par l'utilisation d'une architecture qui réduit à quatre le nombre de cycles d'instruction nécessaires pour les opérations de « butterfly update » de Viterbi pour le décodage d'un canal GSM. Les autres caractéristiques du processeur TMS320C54x sont :

- ✓ quatre bus internes et générateurs d'adresses doubles permettent le traitement des opérandes multiples pour réduire des goulots d'étranglement de mémoire,
- ✓ un additionneur de 40 bits et deux accumulateurs de 40 bits pour exécuter dans un cycle d'horloge des instructions parallèles,
- ✓ un deuxième additionneur de 40 bits à la sortie du multiplicateur pour permettre des opérations MAC (Multiplier Accumulator) non pipeliné, ainsi que une addition et une multiplication en parallèle,

- ✓ normalisation et codage exponentiel dans un seul cycle d'horloge pour des arithmétiques à virgule flottante,
- ✓ des nouvelles instructions d'un seul cycle pour exécuter efficacement des tâches communes DSM comme un filtre symétrique de FIR,
- ✓ une unité arithmétique et logique (UAL) de 40 bits avec un dispositif de configuration pour permettre l'exécution des opérations doubles ou d'un cycle d'horloge.

Les processeurs DSP de la famille TMS320C54x existent en plusieurs configurations pour répondre à toutes les conditions de traitement et besoin de mémoire pour les applications de multimédia sans fil. Le plus récent processeur TMS320C5410 comporte 64Kbytes de mémoire RAM et 32Kbytes de ROM intégrées dans le circuit. En plus des mémoires RAM internes à la puce, le C5410 comporte également un contrôleur DMA de six canaux.

3.1.4. Processeur d'usage personnalisé

Le processeur XTENSA de Tensilica. Le processeur XTENSA [TEN02] est destiné à être intégré dans des systèmes sur puce, la programmabilité du cœur de ce processeur autorise la personnalisation du circuit et l'ajout de fonctions et d'instructions, par rapport aux besoins de l'application sous développement. Le processeur de base utilise environ 27 500 portes et peut atteindre une fréquence d'horloge jusqu'à 350 MHz. Le processeur Xtensa, en technologie de fabrication 0,13 µm avec une configuration minimale, ne consomme que 0,05 mW/MHz.

Le coeur configurable, extensible et synthétisable du processeur Xtensa fait la différence par rapport aux autres processeurs conventionnels. En utilisant une technologie développée par la société Tensilica, le concepteur d'un système monopuce peut générer un ou plusieurs processeurs Xtensa qui correspondent exactement aux besoins et aux tâches de son application. Le concepteur peut choisir et configurer des caractéristiques prédéfinies de processeur en utilisant la méthodologie de Tensilica Instruction Extension (TIE), qui rajoute une description, similaire aux descriptions Verilog, des datapaths d'exécution, des ports d'entrées/sorties et les registres pour fournir des caractéristiques de performance, de surface et de puissance équivalentes aux circuits logiques câblés. Le concepteur peut utiliser aussi le compilateur XPRES pour analyser l'algorithme C et suggérer automatiquement des options de configuration et des extensions qui vont exécuter un algorithme plus rapidement. Comparé à la conception traditionnelle de matériel, les processeurs Xtensa fournissent la même performance mais avec des avantages supplémentaires comme une réduction du temps de conception.

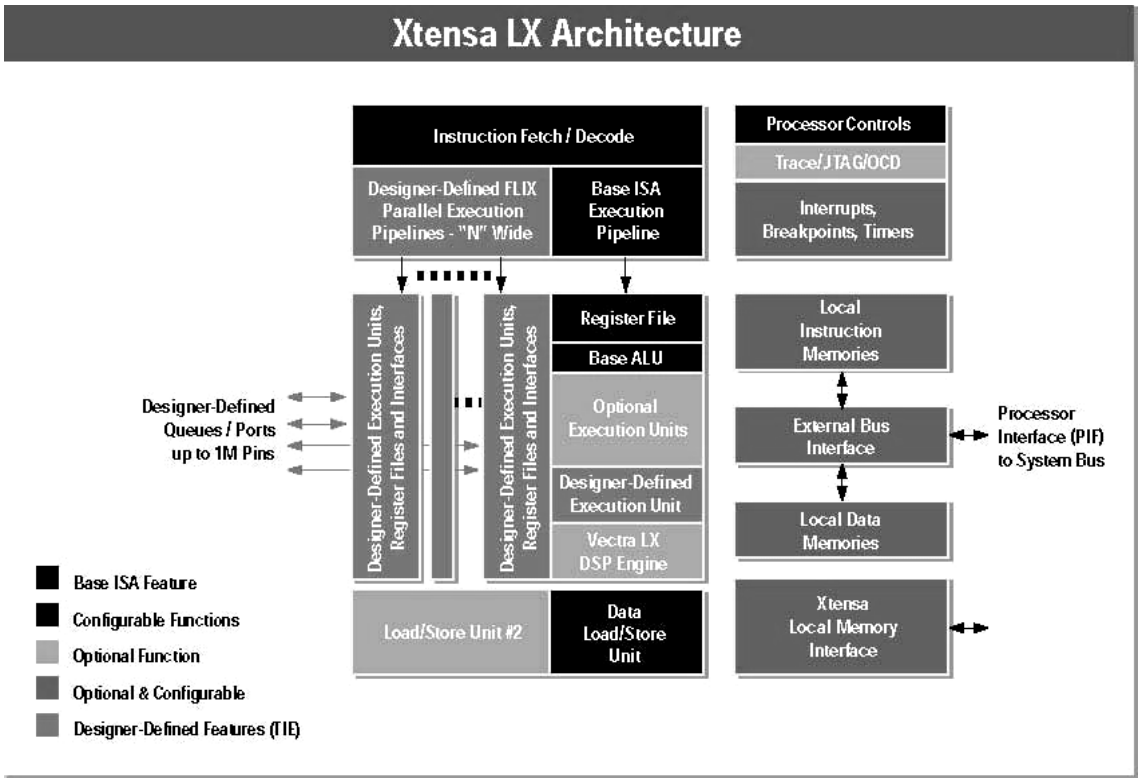


Figure 2.5 : L'architecture du processeur Xtensa LX de la société Tensilica.

L'architecture de processeur Xtensa LX comporte un jeu d'instructions de 32 bits compact optimisé pour des systèmes embarqués. L'architecture de base contient une UAL de 32 bits, jusqu'à 64 registres d'usage universel, 6 registres spéciaux et 80 instructions de base, y compris des instructions RISC de longueur variable (16, 24 ou 32 bits). Enfin, on peut opter aussi pour un pipeline de 5 à 7 étages pour l'accès mémoire. Le processeur Xtensa LX utilise une propre architecture ISA Xtensa, qui permet des réductions significatives de nombre d'instructions comparées par rapport aux autres processeurs RISC. La réduction de la taille du code en nombre d'instructions augmente la performance et améliore la consommation de la puissance, qui est la clé pour baisser le coût dans la conception des systèmes sur puce fortement intégrées.

3.2. Les composants de mémorisation

Une mémoire est un circuit à semi-conducteur permettant d'enregistrer, de conserver et de restituer des informations (instructions et variables). C'est cette capacité de mémorisation qui explique la polyvalence des systèmes numériques et leur adaptabilité aux nombreuses applications. Les caractéristiques principales d'une mémoire sont :

- ✓ la capacité de la mémoire : c'est le nombre total de bits que contient la mémoire,
- ✓ le format des données : c'est la largeur du mot mémorisable,

- ✓ le temps d'accès : c'est le temps qui s'écoule entre l'instant où a été lancée une demande de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données,
- ✓ le temps de cycle : il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture,
- ✓ le débit : c'est le nombre maximum d'informations lues ou écrites par seconde,
- ✓ volatilité : elle caractérise la permanence des informations dans la mémoire.

La conception et le fonctionnement des mémoires est largement documentée, et n'est pas inclus dans ce mémoire pour brièveté. Au lieu de cela, nous allons nous concentrer sur les mémoires actuellement utilisées dans la conception des systèmes sur puce, décrivant certaines des innovations et des limitations technologiques qui sont inhérentes à l'utilisation des mémoires. Nous allons discuter seulement des types des mémoires que nous avons rencontrées déjà implémentées dans des systèmes monopuce réels commercialisés comme par exemple [SON02] et [HIT97].

3.2.1. Notion de hiérarchie mémoire

Une mémoire idéale serait une mémoire de grande capacité, capable de stocker un maximum d'informations et possédant un temps d'accès très faible afin de pouvoir travailler rapidement sur ces informations. Mais il se trouve que les mémoires de grande capacité sont souvent très lentes et que les mémoires rapides sont très chères. Et pourtant, la vitesse d'accès à la mémoire conditionne dans une large mesure les performances d'un système. En effet, c'est là que se trouve le *goulot d'étranglement* entre un processeur capable de traiter des informations très rapidement et une mémoire beaucoup plus lente (par exemple : un processeur actuel à 3Ghz et une mémoire à 400MHz). Or, on n'a jamais besoin de toutes les informations au même moment. Afin d'obtenir le meilleur compromis coût/performance, on définit donc une hiérarchie mémoire. On utilise des mémoires de faible capacité mais très rapides pour stocker les informations dont le processeur se sert le plus et on utilise des mémoires de capacité importante mais beaucoup plus lentes pour stocker les informations dont le processeur se sert le moins. Ainsi, plus on s'éloigne du processeur et plus la capacité et le temps d'accès des mémoires vont augmenter.

- ✓ Les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- ✓ La mémoire cache est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- ✓ Les mémoires collées au processeur ou TCM (Tightly Coupled Memory) sont apparues récemment dans la hiérarchie de stockage des données des systèmes monopuce. Ces

mémoires très rapides offrent une plus grande capacité de mémorisation qui peut être accédé dans un seul cycle d'horloge.

- ✓ La mémoire principale est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.

Il est possible de rencontrer au sein d'une architecture monopuce, les technologies de mémorisation suivantes :

- ✓ des unités de mémoire de masse, telles que des blocs FLASH ou FRAM, contentant des données volumineuses et faiblement volatiles
- ✓ des unités de mémoire morte, de faible capacités, contentant les instructions de programmes dédiés aux premiers étapes de la mise en fonction de l'application, ainsi qu'à sa mise à jour par réprogrammation de mémoires FLASH
- ✓ des blocs de mémoires vive DRAM ou SDRAM pour les données volatiles
- ✓ des petits blocs cache composés de mémoires RAM de faible capacité et de contrôleurs matériels en charge de maintenir régulièrement une copie des instructions et/ou données accédée ou voisines des derniers accès. Cette technique permet de bénéficier de la rapidité des SRAM, sans être pénalisé par leur encombrement.

3.2.2. Les mémoires statiques SRAM

Les spécifications d'un système déterminent le type de mémoire qui convient le mieux à une application particulière. Avec l'apparition des systèmes monopuce, intégrant tous les composants sur une seule puce, le choix du type et de la technologie de la mémoire est devenu crucial. Car la performance d'un système monopuce est directement influencée par les caractéristiques des mémoires utilisées comme la capacité de stockage, le temps d'accès et le débit d'information. Nous allons faire une présentation des différentes mémoires SRAM [ETZ03] utilisées couramment, permettant d'augmenter la bande passante et la performance d'un système. La mémoire SRAM est plus souvent la solution préférée des ingénieurs pour l'implémentation dans les systèmes monopuce, malgré la surface importante de silicium utilisé. Leur rapidité et basse consommation, sont les avantages principaux, par rapport aux mémoires dynamiques, quand la plus grande partie des systèmes monopuce sont dédiés pour des domaines d'application utilisant des piles.

SSRAM : La mémoire statique synchrone est un composant qui contient un seul port d'accès à la matrice mémoire, constituée d'un bus commun d'entrées/sorties bidirectionnel. Il est possible d'effectuer soit une fonction d'écriture, soit une fonction de lecture à un instant donné, mais pas les deux. Les SRAM synchrones peuvent être directes ou à pipeline.

Contrairement à un circuit direct, la mémoire à pipeline dispose d'un registre de sortie. Le registre de sortie permet à un circuit à pipeline d'atteindre des fréquences d'horloge plus élevées qu'un circuit direct, mais au détriment de la latence. Cette dernière correspond au délai qui existe entre la lecture des données en mémoire et leur disponibilité en sortie (habituellement exprimée en cycles d'horloge). Généralement, les systèmes réseau nécessitent une bande passante ou un débit de données élevé. La latence initiale peut donc être sacrifiée au bénéfice d'un plus grand débit offert par un circuit à pipeline.

SSRAM ZBT : Une mémoire avec un temps de retournement nul ou ZBT (Zero Bus Turnaround) est optimisée pour réduire la latence entre des cycles alternés de lecture et d'écriture. Comme une mémoire statique synchrone, elle comporte un seul bus commun d'entrées/sorties bidirectionnel. Ce qui distingue cette mémoire d'une mémoire statique synchrone standard : les circuits logiques internes sont utilisés pour gérer le flux de données pipeline et pour minimiser la latence entre des opérations successives de lecture et d'écriture. Cette architecture convient très bien aux applications qui sont partagées à 50 % entre les opérations de lecture et les opérations d'écriture. Cependant, lors de l'envoi de longs flux de données au circuit, sa bande passante optimum sera équivalentes à celle d'une SRAM synchrone à pipeline.

SRAM à entrées/sorties séparées : Comme leur nom l'indique, ces circuits fractionnent le port commun d'entrées/sorties en des bus d'entrée et de sortie séparés pour éviter les conflits et les retournements de bus. Toutefois, comme pour une SRAM à port commun entrées/sorties, un seul accès de lecture ou d'écriture peut avoir lieu pour n'importe quel cycle d'horloge donné.

SRAM DDR : Une mémoire double débit de données ou DDR (Double Data Rate) utilise un seul bus commun d'entrées/sorties bidirectionnel. Contrairement à une SRAM synchrone, un circuit DDR nécessite une horloge différentielle ou deux horloges en opposition de phase (C et /C). Les données sont verrouillées au niveau du réseau mémoire à chaque front montant de C ou /C. Le transfert de données et la vitesse d'entrées/sorties sont alors doublés à deux fois par la fréquence d'horloge.

SRAM QDR : La mémoire statique quadruple débit de données ou QDR (Quad Data Rate) a été développée spécialement pour les applications de réseaux hautes vitesses. L'architecture de base combine les avantages des mémoires à E/S séparées et des mémoires DDR. Ce circuit possède deux ports séparés (un pour les entrées et l'autre pour les sorties) séquencés par deux entrées d'horloge en opposition de phase. Chaque port est adressé indépendamment de l'autre et permet des accès simultanés à la matrice mémoire. La RAM QDR permet d'obtenir une bande passante quatre fois plus grande qu'une SRAM synchrone. Elle peut accepter des horloges déphasées entre

les entrées et les sorties pour tenir compte des délais de propagation externes, mais par souci de simplicité elle sera représentée par un couple d'horloges unique (C et /C).

SRAM double port : Les mémoires double port permettent de réaliser une multitude d'applications qui ne sont pas possibles avec les précédents circuits. Elles comportent deux bus communs d'entrées/sorties bidirectionnels et elles supportent aussi des domaines d'horloge indépendants sur chaque port. Ceci permet à la mémoire double port de devenir une passerelle (ainsi qu'un buffer) entre des circuits appartenant à des domaines d'horloge différents et de synchroniser le flux de données entre eux. Elle permet aussi d'optimiser la bande passante entre les circuits d'interface en permettant le transfert de données à la vitesse optimum du circuit.

Il existe actuellement une grande diversité d'architectures SoC basées sur les mémoires statiques. Dans notre expérimentation nous avons aussi travaillé et implémenté des mémoires statiques synchrones.

3.2.3. Les mémoires dynamiques DRAM

Dans les mémoires vives dynamiques ou DRAM (Dynamic Random Access Memory), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur. L'avantage de cette technique est qu'elle permet une plus grande densité d'intégration, ce qui entraîne une réduction de la consommation d'énergie aussi. L'inconvénient principal des mémoires dynamiques est la présence de courants de fuite dans le condensateur qui contribue à sa décharge. Ainsi, l'information est perdue si on ne la recharge pas périodiquement. Les mémoires dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger. Ce qui mène à la diminution du débit ou la bande passante de la mémoire

Du fait de leur haute intégration un grand nombre d'innovations au cours du temps ont été mises en jeu pour améliorer leurs performances et le débit des mémoires dynamiques comme le rafraîchissement, la pagination et les différentes architectures par exemple RAMBUS [RAM00]. Malgré tous ces différents mécanismes, les mémoires dynamiques sont restées en arrière-plan dans la conception des systèmes sur puce.

3.2.4. Les mémoires mortes (ROM)

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou ROM (Read Only Memory). Suivant le type, la méthode de programmation, il existe plusieurs types de ROM. Nous allons discuter seulement la mémoire FLASH EPROM car c'est la mémoire qui a connu un essor très important ces dernières années dans la conception des systèmes

sur puce avec le boom de la téléphonie portable et des appareils multimédia (PDA, appareil photo numérique, lecteur MP3, etc.).

FLASH EPROM : La mémoire Flash s'apparente à la technologie de la mémoire EEPROM. Elle est programmable et effaçable électriquement. Il existe deux technologies différentes qui se différencient par l'organisation de leurs réseaux mémoire : l'architecture NOR et NAND. L'architecture NOR propose un assemblage des cellules élémentaires de mémorisation en parallèle avec les lignes de sélection comme dans une EEPROM classique. L'architecture NAND propose un assemblage en série de ces mêmes cellules avec les lignes de sélection. D'un point de vue pratique, la différence majeure entre NOR et NAND tient à leurs interfaces. Alors qu'une NOR dispose de bus d'adresses et de données dédiés, la NAND est dotée d'une interface d'entrées/sorties indirecte. Par contre, la structure NAND autorise une implantation plus dense grâce à une taille de cellule approximativement 40 % plus petite que la structure NOR.

Si NOR et NAND exploitent toutes deux le même principe de stockage de charges dans la grille flottante d'un transistor, l'organisation de leur réseau mémoire n'offre pas la même souplesse d'utilisation. Les Flash NOR autorise un adressage aléatoire qui permet de la programmer octet par octet alors que la Flash NAND autorise un accès séquentiel aux données et permettra seulement une programmation par secteur comme sur un disque dur.

3.3. Les interfaces d'entrée/sortie

Les interfaces d'entrées/sorties sont très importantes pour les performances du système. Rien ne sert d'avoir un processeur calculant très rapidement s'il doit souvent perdre son temps pour lire des données ou écrire ses résultats. Durant une opération d'entrée/sortie, l'information est échangée entre la mémoire principale et un périphérique relié au système. Cet échange nécessite une *interface* ou *contrôleur* pour gérer la connexion.

Les fonctions de ces composants de communication sont très élémentaires, mais elles influent directement sur les performances de l'ensemble de l'application. Alors il faut que les composants de communication soient flexibles et adaptables pour les différents schémas d'interconnexions, ce qui nécessite de les considérer sous trois angles :

- ✓ l'extensibilité ou l'aptitude du composant à supporter un nombre donné d'utilisateurs concurrents sans altérer ni ses fonctionnalités ni ses performances globales,
- ✓ La compatibilité aux schémas de communication utilisés par l'application : point à point, multipoint, transfert en rafale, transfert non préemptif,
- ✓ La compatibilité avec d'autres réseaux éventuellement de nature différente.

Plusieurs composants sont employés pour répondre aux critères et pour effectuer ces échanges.

3.3.1. Les ponts

Les ponts sont une application spécifique des adaptateurs à la connexion de plusieurs bus supportant chacun plusieurs maître. Ils doivent donc s'enquérir de :

- ✓ l'attribution du bus. En termes d'organisation, l'initiateur de la communication ou maître prend le contrôle de son bus et accède au pont se comportant alors comme un esclave. Le pont formule alors une requête pour obtenir l'accès au bus connecté à l'esclave et agit ainsi en tant que maître sur ce bus,
- ✓ La transformation de protocole dans le cas de bus hétérogènes, Toute donnée échangée par un maître sur son bus suivant un protocole d'échange qui peut ne pas être supporté par le bus connecté à l'esclave cible. Le pont doit alors adapter ces deux protocoles.

Ils sont implémentés en logique câblée. Leur utilisation est requise lorsque l'adaptation entre deux réseaux de communication doit être opérée. Cet état de fait se présente lorsque le réseau est hiérarchisé afin de :

- ✓ offrir localement les performances requises,
- ✓ accentuer le parallélisme en utilisant des ressources de transports concurrents,
- ✓ reporter les problèmes d'incompatibilité de protocoles des différents ensembles processeur/bus natif, sur un unique composant et permettre aussi loin que possible l'utilisation des protocoles natifs.

3.3.2. Le contrôleur de DMA

Un circuit appelé contrôleur d'accès direct à la mémoire ou DMA (Direct Memory Access) prend en charge les différentes opérations et permet le transfert de blocs de données entre la mémoire et un périphérique sans passer par le processeur.

Le contrôleur de DMA se charge entièrement du transfert d'un bloc de données. Le processeur doit tout de même :

- ✓ initialiser l'échange en donnant au DMA l'identification du périphérique concerné,
- ✓ donner le sens du transfert,
- ✓ fournir l'adresse du premier et du dernier mot en mémoire concernés par le transfert.

Un contrôleur de DMA est doté d'un registre d'adresse, d'un registre de donnée, d'un compteur et d'un dispositif de commande (logique câblée). Pour chaque mot échangé, le contrôleur de DMA demande au processeur le contrôle du bus, effectue la lecture ou l'écriture mémoire à l'adresse contenue dans son registre et libère le bus. Il incrémente ensuite cette adresse et

décrémente son compteur. Lorsque le compteur atteint zéro, le dispositif informe le processeur de la fin du transfert par une ligne d'interruption.

Le principal avantage est que pendant toute la durée du transfert, le processeur est libre d'effectuer un traitement quelconque. La seule contrainte est une limitation de ses propres accès mémoire pendant toute la durée de l'opération, puisqu'il doit parfois retarder certains de ses accès pour permettre au dispositif d'accès direct à la mémoire d'effectuer les siens : il y a apparition de vols de cycle.

3.3.3. Les standards des interfaces

Afin de faciliter l'intégration de divers composants au sein d'une même architecture, plusieurs standards de protocoles et interfaces ont été développés pour que lors de la conception de ces composants, ces protocoles soient adoptés. Ces standards, afin de devenir tels, ont été élaborés par des initiatives de collectifs et des consortiums afin de proposer des solutions « ouvertes » dont le coût d'élaboration ne soit pas supporté par une seule compagnie. On trouve ainsi :

- ✓ la spécification Wishbone de Silicore Corporation [SIL04], adoptée par le collectif OpenCores [OC04] pour sa simplicité ;
- ✓ la spécification OCP (Open Core Protocol) développée par Sonics et soutenue par l'OCP-IP (OCP International Partnership) [OCP04] ;
- ✓ le consortium Virtual Socket Interface Alliance (VSIA) [VSI04] définit un ensemble plus large de standards destinés à la réalisation normée complète d'un système sur silicium. Dans le cadre de la conception d'un réseau sur silicium, le concepteur sera concerné par le standard de documentation et de spécification SLIF, un standard d'abstraction des interfaces de bus On-Chip Virtual Component Interface (VCI), et une représentation standardisée des types de données, le System-Level Data Types Standard.

De manière générale, ces spécifications de protocoles définissent une interface point à point qui fournit un ensemble standard de signaux de données de contrôle et de test permettant aux coeurs du système de communiquer entre eux. Ces standards offrent une manipulation et une utilisation simplifiées des composants grâce à leur compatibilité directe. Contrairement aux bus systèmes, ils n'imposent pas de topologie du réseau. Cependant, ils n'en offrent pas non plus. Et ils ne répondent pas toujours pleinement aux besoins spécifiques des applications. Leur respect peut conduire à l'implémentation d'un jeu de fonctionnalités qui ne sont pas toutes utilisées.

Leur utilisation reste faiblement répandue, car les utilisateurs potentiels leur reprochent les points précédemment cités ou alors la stratégie ou la politique des entreprises utilisatrices ne

s'accommode pas des décisions des organismes de standardisation souvent « orientées » par d'autres entreprises. Des solutions propriétaires leur sont donc préférées.

3.4. Les différents types de communication et les composants de communication

La fonction d'un système à base de processeurs, quel qu'il soit, est le traitement de l'information. Il est donc évident qu'il doit acquérir l'information fournie par son environnement et restituer les résultats de ses traitements. Chaque système est donc équipé d'une ou plusieurs interfaces d'entrées/sorties plus un schéma d'interconnexion permettant d'assurer la communication entre le processeur et le monde extérieur.

3.4.1. Communication par bus partagé

Dans un système informatique, les différents sous-systèmes doivent avoir des interfaces pour communiquer. Par exemple, la mémoire et le CPU doivent communiquer, toute comme le CPU et les composants entrées/sorties. Cette communication est réalisée généralement via un bus. Le bus sert de lien de communication partagé par les sous-systèmes.

Le bus doit allouer dynamiquement des ressources de communication à un maître en fonction d'un schéma d'arbitrage. Il existe plusieurs schémas d'arbitrage, parmi les plus utilisés sont :

- ✓ l'arbitrage par priorités dynamiques selon l'ordre d'apparition des requêtes,
- ✓ l'arbitrage en fonction d'une priorité fixe statiquement assignée à chaque maître,
- ✓ l'arbitrage « round robin », l'attribution du bus systématique et cyclique.

Les transferts d'informations sur le bus peuvent être synchronisés par rapport à une horloge globale pour le système, ou par rapport à une horloge locale du processeur. Il existe aussi des bus asynchrones.

La communication par bus partagé est effectuée en trois phases :

- ✓ l'attribution du chemin de données,
- ✓ la synchronisation des extrémités,
- ✓ le routage de l'information.

Les deux avantages essentiels de la structure de bus sont son faible coût et sa souplesse. L'inconvénient principal d'un bus est qu'il crée un goulot d'étranglement capable de limiter le débit maximal d'entrées/sorties. Une des raisons pour lesquelles la conception d'un bus est si difficile est que sa vitesse maximale est limitée surtout par des facteurs physiques : la longueur du bus en nombre de composants et donc charge du bus.

Le bus AMBA (Advanced Multi Masters Bus Architecture) [ARM99], et le STBus [BRI03] sont des exemples des bus partagés utilisés dans les systèmes sur puce. Ils peuvent supporter la connexion de plusieurs composants initiateurs de communication (Multi Master) et doivent donc gérer en temps réel l'attribution des moyens pour la communication entre deux composants.

3.4.1.1. Advanced Microcontroller Bus Architecture de la société ARM

Le bus AMBA [ARM 99] regroupe une famille de bus partagés destinée à satisfaire le plus grand nombre de besoins :

Advanced Peripheral Bus (APB) : ce bus fournit une interface de complexité minimale et n'autorise qu'un seul maître, la plupart du temps le pont de connexion avec un autre bus, multi maîtres nécessairement, et la plupart du temps de la même famille. Ce bus destiné à minimiser la consommation n'autorise qu'un faible bande passante et ne fournit aucun pipeline des transactions.

Advanced High Performance Bus (AHB) : c'est un bus multi maîtres synchrone à large bande passante et haut débit. Il supporte les transactions en rafales, non préemptives et différées, cela avec un nombre élevé de maîtres (jusqu'à 16 par bus). Pour de meilleures performances, les données associées à une adresse sont présentes sur le bus un cycle après l'adresse, ce qui permet un fonctionnement en pipeline. Un chemin de données multiplexé – d'une largeur extensible à 128 bits – est préféré à des lignes de bus trois états. Cela permet une fréquence de fonctionnement plus élevée et simplifie/améliore l'automatisation de la synthèse du système. L'arbitre et le décodeur d'adresses sont centralisés et reliés point à point avec l'ensemble des périphériques, au contraire de AMBA ASB où le maître devait passer par le bus pour demander et obtenir une communication.

La figure 2.6 illustre une architecture type à base de bus hiérarchiques AMBA.

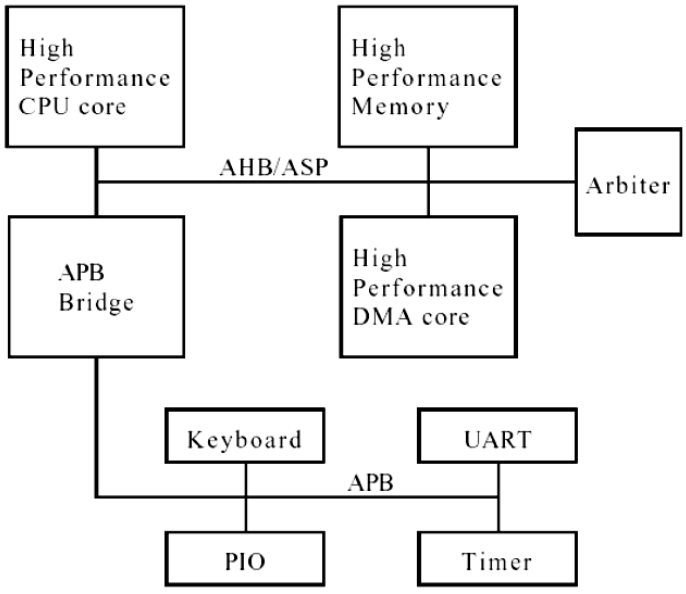


Figure 2.6 : Une architecture type à base de bus hiérarchiques AMBA.

La politique de l'arbitre est laissée à la discrétion de l'implémentation. Elle peut être équitable ou non, avec ou sans famine, selon les besoins de l'application et des éléments du système. Le bus AMBA spécifie un protocole d'accès spécifique efficace, via le bus, pour le test des modules raccordés (mais il ne s'agit pas d'un test du bus en lui-même).

Le bus AHB a été développé pour apporter des solutions à la précédente version du bus AMBA haute performance, le ASB (Advanced System Bus). Le protocole de ce dernier, rencontrait quelques limitations. Pour contourner le problème de son manque d'extensibilité, la largeur du bus et la fréquence d'horloge, ont progressivement augmenté.

Toutefois, un bus trop large ne sera pas plus efficace pour les transactions dont la taille est inférieure à la taille du mot. Si l'interface doit concaténer les transactions pour être efficace, elle perd ses avantages de simplicité et de faible latence.

Même lorsque le débit est suffisant, le bus ne convient plus aux grands systèmes avec de nombreux éléments à raccorder. Le goulot est alors l'arbitre central, qui doit être dimensionné pour élire un maître parmi des dizaines. S'il ne peut plus accomplir cette tâche efficacement (ce qui est inévitable), la latence de service des requêtes s'accroît et une proportion notable des cycles disponibles est perdue dans les délais d'arbitrage [GUE00].

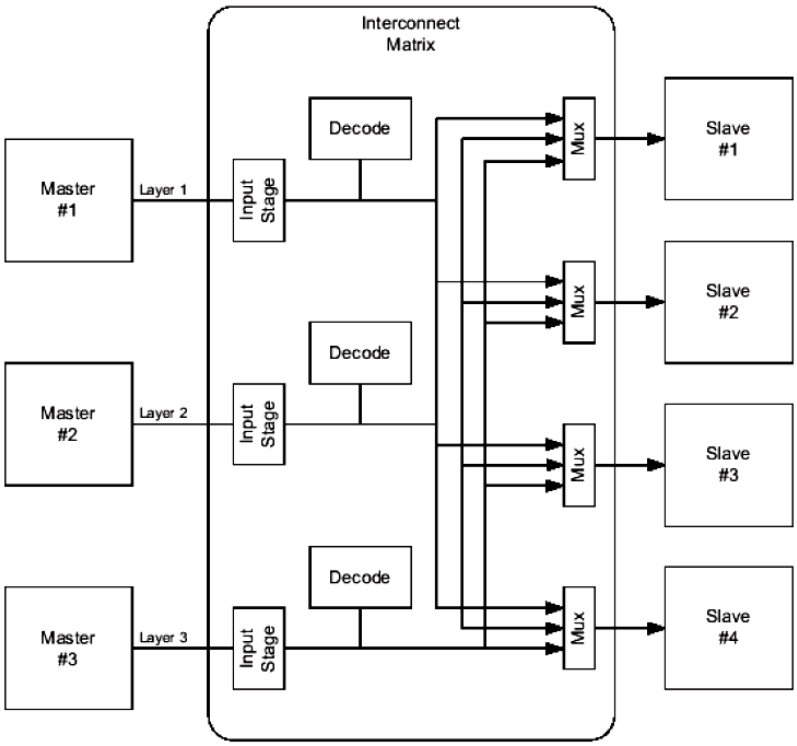


Figure 2.7 : L'exemple de matrice d'interconnexion AHB avec 3 maîtres et 4 esclaves.

Multi-layer AHB [ARM02] : pour contourner l'absence d'extensibilité et finalement assurer la continuité du standard AMBA, ARM offre un réseau d'interconnexion basé sur le protocole AHB pour remplacer le bus centralisé. Il s'agit d'une matrice d'interconnexion qui distribue un bloc de

décodage par maître et un bloc d'arbitrage par esclave. Cette extension a cependant rencontré peu de succès, car elle reste mono horloge et nécessite donc la gestion globale – déjà délicate pour un bus centralisé – sur un réseau distribué. La figure 2.7 illustre un exemple de BusMatrix ou matrice d'interconnexion multi - layer AHB de 3 maîtres et 4 esclaves. Une telle matrice configurée pour interconnecter de 2 maîtres avec 2 esclaves est utilisée dans la suite de ces travaux de thèse.

AHB-Lite [AMB02] : il s'est avéré à l'usage que bon nombre de concepteurs avaient besoin des performances liées au protocole de bus AHB mais avec un usage amélioré des services du protocole APB destiné à un maître unique. Le bus AHB implémentait des fonctionnalités inutiles dans ce cas-là. Le protocole AHB-Lite fournit le jeu de services réduit pour un maître unique tout en conservant les potentialités de transactions sophistiquées : en rafale, rémissibles, etc.

3.4.1.2. Le STBus – STMicroelectronics

Le STBus [BRI03], [COM02] est un environnement permettant la synthèse des communications dans un système sur puce. Il s'agit donc d'un "générateur" de bus d'interconnexion, dont le composant principal est un "nœud" générique permettant à plusieurs initiateurs d'adresser plusieurs cibles. Ces nœuds étant cascadables, cet environnement permet donc de construire un système de bus hiérarchiques. Différents modules de conversion permettent d'interconnecter des bus de largeurs différentes, ou des sous-systèmes possédant des "endianness" différentes.

Les objectifs et les principes de fonctionnement sont très proches de ceux de la norme VCI [VSI04]. Comme VCI, le STBus définit des communications de point à point dans un espace adressable partagé. Il vise à découpler la conception de composants virtuels réutilisables et la conception des interconnexions. Les signaux d'interface définis par le STBus semble respecter aussi bien la syntaxe que la sémantique du standard VCI (particulièrement pour le STBus type 2), et le STBus peut être vu comme une implémentation particulière du standard VCI.

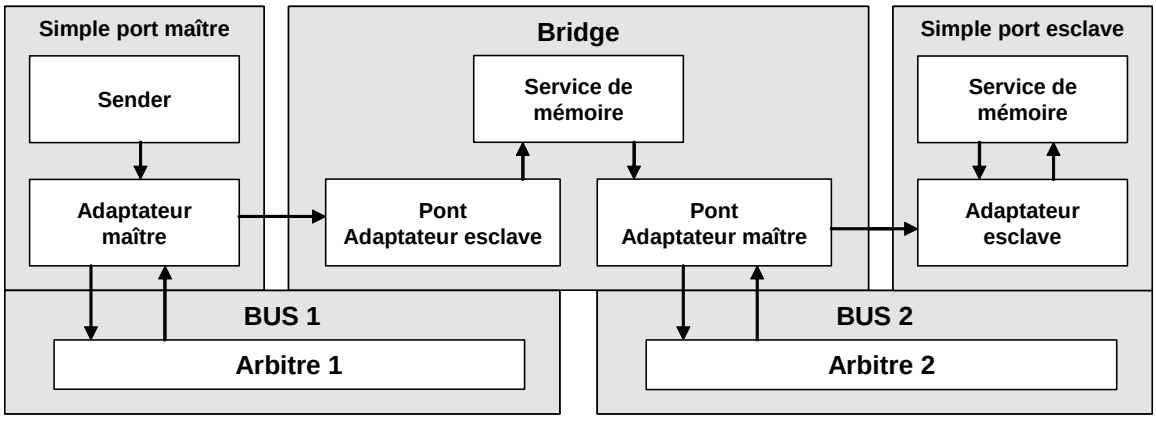


Figure 2.8 : Modèle fonctionnel des services de communication du STBus.

L'avantage de définir ces standards d'interface est de conserver la liberté de choix de la topologie d'interconnexion. Cependant le STBus propose également un choix restreint mais suffisant de topologies en bus centralisé. La topologie en cas combine ici les avantages d'une architecture distribuée (large bande passante, allègement des contentions) avec un standard d'interface conventionnel et relativement flexible. L'interconnexion autorise le pipeline par FIFO aux interfaces des maîtres et esclaves, les services de transactions *split* et *burst*. Chaque noeud peut implémenter son propre algorithme d'arbitrage. La figure 2.8 présente le modèle fonctionnel de l'interaction des services de communication du STBus indépendamment de la topologie choisie.

Le modèle fonctionnel de STBus se compose des modules architecturaux suivants :

Adaptateur maître : relie l'interface du maître avec le protocole de bus qu'il emploie et il contient la majeure partie du protocole de STBus. Il définit aussi la fonction de demande d'accès au bus pour une transaction bloquante (sans une canalisation (pipeline) comme par exemple pour un transfert d'un processeur), ou pour une transaction non bloquante (où un FIFO est nécessaire pour canaliser (pipeline) le transfert comme il s'agit d'un composant ASIC). Chacune des transactions est coupée en paquets de taille paramétrable. Il faut aussi configurer les paramètres de STBus comme par exemple la largeur du bus, l'horloge et le type et la taille des FIFO.

Adaptateur esclave : connecte l'interface d'esclave avec le bus. Chacun des paquets reçus est mis dans un FIFO et transmis au service de mémoire. Quand la transaction est finie, ses cellules correspondantes dans l'adaptateur de maître sont libérées et l'arbitre est relâché. Puis la prochaine transaction stockée dans le FIFO maître est envoyée. Quand le FIFO cible est plein, l'arbitre est bloqué jusqu'à ce qu'une cellule soit libérée. Ce service a les mêmes paramètres que l'adaptateur du maître.

Arbitre : L'arbitre gère la possession du bus et le transfert des données à partir de la source vers la destination. Deux schémas d'arbitrage sont implémentés : une arbitrage par priorité programmable et une arbitrage utilisant des algorithmes de « Less Recently Used ».

Ponts : Ils ont été développés pour permettre l'interconnexion du STBus avec d'autres types de bus.

Un des avantages du STBus est que sa conception est appuyée par des outils de traces et d'analyse de performance et des outils d'exploration des paramètres du réseau d'interconnexion (topologie, arbitrage, FIFO, etc.), en particulier l'outil *Virtual Component Codesign* (*VCC*) de Cadence [VCC03] qui permet l'exploration d'architectures et le partitionnement de logiciel et de matériel [BAG02]. En outre la bibliothèque de modules constituants le STBus fournit des ponts vers les autres modèles de réseaux d'interconnexion, en particulier vers les systèmes ARM et Tensilica.

3.4.2. Communication à la commutation

Dans les dernières années, un nouveau type de communication dans les circuits monopuce est apparu : les réseaux de communication (Network on Chip). Pour des raisons d'efficacité, on essaie de plus en plus de connecter des sous-systèmes d'une application monopuce indépendants entre eux par l'intermédiaire d'un réseau. On permet ainsi aux applications de partager et d'échanger les mêmes informations.

Pour faire circuler une information sur un réseau, on peut utiliser principalement deux stratégies. Soit l'information est envoyée de façon complète, soit elle est décomposées en petits morceaux (paquets). Les paquets sont alors envoyés séparément sur le réseau puis rassemblés par la machine destinataire. On parle souvent de réseaux à *commutations de paquets*.

La première stratégie n'est que très rarement utilisés car les risques d'erreurs sont trop importants. Les règles et les moyens mis en œuvre dans l'interconnexion et le dialogue des systèmes définissent le protocole et l'architecture du réseau. Toutes ces règles sont définies par des normes pour que les différents systèmes puissent communiquer entre eux.

Beaucoup de travaux se sont concentrés sur le développement d'un nouveau type de réseau de communication mieux adapté aux applications multiprocesseur. Les réseaux proposés sont tous attractifs et sont bien adaptés pour certains domaines d'application.

3.4.2.1. Le réseau SPIN de l'Université Paris VI

C'était la première proposition complète de réseau sur puce à commutation de paquets. Il possède une topologie en arbre élargi, ce qui le rend extensible par nature et performant de par sa redondance de liens. Le routage des paquets dans le réseau peut être statique ou dynamique (suivant si la table de commutation évolue au cours du temps). Le routage dynamique est plus résistant au problème de congestion, mais nécessite un réordonnancement des paquets à la réception. La couche transport de ce réseau, matérialisée par le protocole RFW7 décrit dans [SPI00], implémente donc deux services : La gestion de flux du bout en bout et le réordonnancement des paquets à la réception. La gestion de flux dans SPIN est basée sur l'émetteur et repose sur le principe du crédit d'émission. Ce principe simple permet de gérer la congestion du réseau. L'émetteur possède un compteur (le crédit d'émission) qui est décrémenté à chaque envoi d'un paquet. Le récepteur, en fonction du remplissage de son tampon, envoie ou non un message à l'émetteur pour que celui-ci incrémente son crédit. Ainsi si le récepteur est bloqué (débordement du tampon), l'émetteur va rapidement s'arrêter d'émettre, évitant d'envoyer des paquets qui vont stagner dans le réseau. SPIN doit aussi fournir un système de réordonnancement. Pour cela les données ne sont pas écrites dans le tampon dans leur ordre d'arrivée (cas d'une file classique), mais selon leur position dans le paquet. L'écriture est cyclique dans le tampon. Ainsi un pointeur

de lecture tourne en permanence, s'arrêtant si un paquet n'est pas encore arrivé, et un pointeur d'écriture écrit à la bonne place dans le tampon. La gestion des débordements (si le pointeur d'écriture dépasse le pointeur de lecture) est réalisé par un principe simple : les paquets ne pouvant pas être écrits sont renvoyés dans le réseau.

La taille du tampon de réception est de manière évidente un paramètre critique pour les performances du réseau. Il ne peut pas être trop grand pour des raisons de surface et de consommation, ni trop petit pour limiter les débordements.

3.4.2.2. Le réseau Octagon de STMicroelectronics

La motivation pour ce travail vient d'un besoin d'architecture de communication sur puce haute performance afin d'aider à fournir la capacité de traitement exigé par les versions les plus récentes des routeurs. Les architectures traditionnelles d'interconnexion telles que le bus partagé et les crossbars auront des difficultés à répondre à ces besoins de débits tout en maintenant des coûts raisonnables.

L'unité de base de Octagon [KAR02] se compose de huit nœuds et de douze liens bidirectionnels connectés comme représentés sur la figure 2.9.

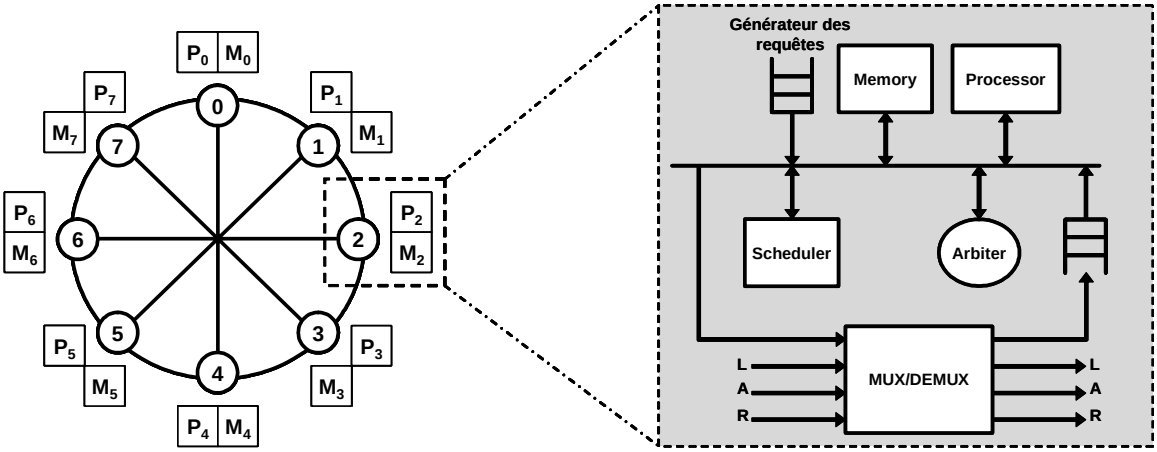


Figure 2.9 : L'exemple de l'architecture et d'un nœud du réseau Octagon.

La figure 2.9 à droite montre des détails de modèle de chacun des noeuds du réseau Octagon. En addition avec le générateur de requêtes, le processeur et la mémoire, chaque noeud possède trois ports bidirectionnels d'entrée et de sortie, marqué par gauche (L), à travers (A), ou droite (R), conformé avec son voisin associé. La fonctionnalité logique du noeud émule un simple commutateur 4x4 non bloquant (plus le processeur et la mémoire). D'autres particularités du nœud du réseau Octagon sont que le commutateur ne possède aucune mémorisation (buffering) des données ni à l'entrée ni à la sortie du nœud et que l'arbitrage de commutateur est réalisé par un scheduleur central.

L'architecture Octagon a les propriétés suivantes :

- ✓ La communication entre n'importe quelle paire de nœuds peut être exécutée soit directement, soit avec un nœud intermédiaire.
- ✓ Débit total plus élevé qu'un bus partagé ou un crossbar
- ✓ Algorithme simple de routage de plus court chemin
- ✓ Moins de câblage comparé à celui d'une interconnexion crossbar.

Cette vue fournit une deuxième option d'extensibilité pour intégrer un nombre plus grand de composants sur la puce. Octagon peut fonctionner selon les deux modes de commutation : commutation de paquet ou commutation de circuits.

3.4.2.3. Le réseau Æthereal

Philips Research est en cours de conception d'un réseau sur puce proposant la mise en place de qualité de service appelé Æthereal [GEB05]. En effet, ce réseau est très orienté temps réel et multimédia. Elle utilise des connexions paramétrées : lorsqu'un composant veut communiquer avec une autre composant, leurs interfaces respectives initient une connexion qui peut être simple ou multicast et possède des caractéristiques spécifiques (réservation de bande passante, latence bornée, etc.). A l'intérieur de cette connexion, la communication est faite par des transactions contenant des messages prédéfinis (Command, Data, etc.). Ainsi chaque communication peut avoir des caractéristiques différentes correspondant à ses besoins. Le protocole est donc très adaptatif ce qui est un très grand atout. Ce réseau est néanmoins dédié à des applications pour lesquels la surcharge due à l'implémentation de ces services est négligeable devant le gain en qualité globale du SOC.

3.4.3. Les connexions point à point

Il s'agit de l'utilisation de ressources de transmission spécifiques et dédiées à l'implémentation d'un lien entre deux nœuds de calcul.

Le principal usage de ce schéma de communication prend place dans l'implémentation des communications entre un processeur et un accélérateur. Plus particulièrement lorsque l'application est orientée traitement de données et nécessite des latences faibles et de débits importants qu'une ressource partagée ne peut pas offrir. Les architectures cascade pipeline en sont très friandes.

La ressource de transmission étant dédiée à une unique communication, elle est toujours disponible pour cette dernière et peut ainsi lui offrir sa pleine bande passante. De plus du fait de la simplicité du contrôle de telles connexions les débits y sont élevés et les latences quasi-inexistantes.

4. Les composants logiciels de base

Dans les systèmes monopuce le logiciel prend une place très importante. Dans cette section, nous allons tenter de définir les composants de base du logiciel embarqué d'un système monopuce. On en aura besoin de cette définition dans la partie décrivant l'application DivX. On n'a pas eu pour but de définir de manière complète les composants logiciels de base, mais plutôt d'éviter les confusions avec d'autre définitions.

Généralement le logiciel embarqué peut être découpé en quelques couches [JER04] :

- ✓ La couche la plus basse contient les pilotes des contrôleurs d'entrées/sorties et d'autres contrôleurs de bas niveau permettant de commander le matériel. Le code correspondant à cette couche est intimement lié au matériel. Cette couche permet d'isoler le matériel du reste du logiciel et elle appelé couche d'abstraction matériel HAL (Hardware Abstraction Layer) ou logiciel dépendant du matériel HdS (Hardware dependant Software) ;
- ✓ Une couche de gestion de ressources permet d'adapter l'application qu'il faut aussi personnaliser pour l'architecture considérée pour des raisons d'efficacité. Cette couche permet d'isoler l'application de l'architecture. Elle est généralement appelée système d'exploitation ou OS. Souvent on considère que le HAL fait partie de l'OS, mais dans notre travail il est plus pratique de les séparer.
- ✓ Le programme de l'application est la dernière couche dans le logiciel embarqué ;

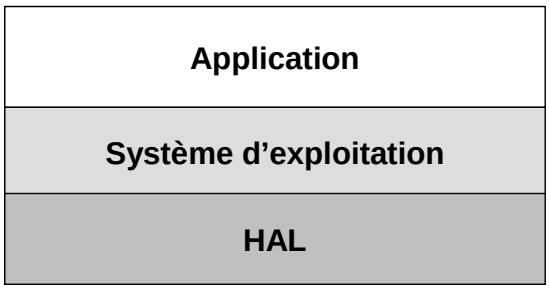


Figure 2.10 : La représentation du logiciel embarqué comme un ensemble des couches.

On peut simplement conclure que la couche d'application représente la partie spécifique de la fonction du circuit et que la couche HAL, la partie spécifique au matériel utilisé dans le circuit, tandis que le système d'exploitation reste « générique » et il peut être réutilisé d'un système sur puce à l'autre.

4.1. La couche d'abstraction matériel ou HAL (Hardware Abstraction Layer)

La couche d'abstraction du matériel ou HAL (Hardware Abstraction Layer) [DSD04] dans la hiérarchie des composants logiciels représente l'interface entre le matériel du système et le système d'exploitation, fournissant une plateforme matérielle abstraite pour exécuter l'application. La couche HAL, ou aussi logiciel dépendant du matériel (HdS : Hardware dependant Software), regroupe tous les services représentants les gestionnaires de périphériques, pour gérer, d'une manière efficace, les ressources matérielles entre les différentes applications. Quand on utilise une couche HAL, l'application n'accède pas directement au matériel mais en employant cette couche d'abstraction. Les HAL donnent la possibilité aux applications d'être indépendantes car ces couches abstraient les informations de ces systèmes comme des mémoires caches, des bus d'entrées/sorties et des interruptions et elles utilisent ces données pour donner aux logiciels la possibilité d'interagir avec les contraintes spécifiques du matériel sur lequel elles s'exécutent.

Le logiciel dépendant du matériel peut posséder de nombreuses sous-familles, dont une sous-couche pour les services liés aux communications de bas niveau, qui gère étroitement les périphériques matériel (IO, mémoire, etc.) et l'amorce (boot) regroupant tous les services liés au démarrage du système qui initialisent les registres du processeur, la table des vecteurs d'interruptions, les espaces de pile, l'espaces d'adressage, etc. Elle charge aussi le noyau du système d'exploitation en mémoire en copiant le code du noyau de la mémoire ROM où il est emmagasiné à la mémoire RAM.

4.2. Le système d'exploitation

De manière pratique, le système d'exploitation est le composant de logiciel le plus important d'un système puisqu'il fournit :

- ✓ une gestion des ressources : processeurs, mémoires, horloges, périphériques, communication interprocessus et inter machines,
- ✓ une base pour l'exécution du programme d'application,
- ✓ une facilité pendant le développement de l'application.

Bref un système d'exploitation assure la gestion des ressources d'un système et de ses périphériques. Le système d'exploitation a pour but :

- ✓ de décharger le programmeur d'une tâche de programmation énorme et fastidieuse et de lui permettre de se concentrer sur l'écriture de son application,
- ✓ de protéger le système et ses usagers de fausses manipulations,
- ✓ d'offrir une vue simple, uniforme et cohérente de la machine et de ses ressources.

4.2.1. Organisation du système d'exploitation

Généralement un système d'exploitation peut être vu comme un ensemble de couches de services (un service étant une unité fonctionnelle élémentaire) [JER04] :

- ✓ une couche de programmation API,
- ✓ et une couche des services de base du système.

Pour chaque couche de service plusieurs réalisations sont possibles.

Les services composant le système d'exploitation peuvent être spécifiques ou définis pour le processeur. Pour les services spécifiques au processeur, leurs réalisations différentes d'un processeur à un autre (par exemple la réalisation est en code assembleur). La réalisation d'un service qui n'est pas spécifique à un processeur reste la même pour tous les processeurs

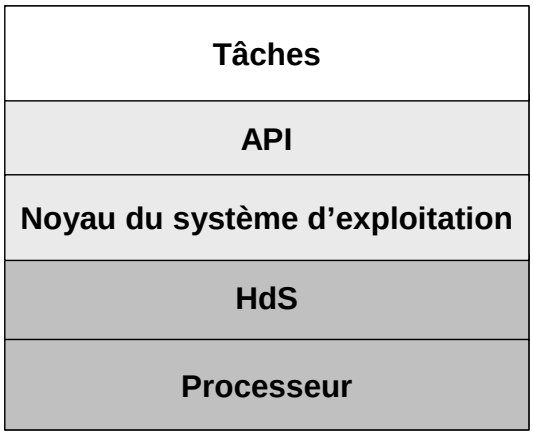


Figure 2.11 : Représentation du système d'exploitation comme un ensemble de couche de services.

La réalisation du système d'exploitation consiste en fait dans le cas de notre approche des systèmes monopuce en :

- ✓ une couche API (Application Programming Interface) représente les appels système de haut niveau invoqués par les tâches logicielles de l'application.
- ✓ un noyau qui représente le cœur du système d'exploitation. Le noyau offre des fonctionnalités de base pour faire tourner les applications utilisateurs dans un système. Il contient un nombre des sous services suivants :
  - le changement de contexte : regroupe tous les services liés à la gestion des contextes associés aux tâches (c'est-à-dire la sauvegarde des registres d'état de la tâche à suspendre et le chargement des registres d'état de la nouvelle tâche prête à être exécutée)
  - l'ordonnanceur : regroupe tous les services liés à l'ordonnancement des tâches. Pour cela, il utilise un algorithme de gestion, généralement par priorité ou tourniquet et gère l'ordre d'exécution des tâches

- la tâche : regroupe tous les services liés à la gestion des tâches. En pratique cette famille de services fait le lien entre les autres sous-familles de la famille noyau (kernel). Elle décrit la structure de la tâche et contient les tables de tâches.

On trouve ensuite des familles de services internes au système d'exploitation qui sont les suivantes :

- ✓ Donnée : regroupe tous le services liés aux structures de données de communication interne du noyau (comme des FIFO).
- ✓ Interruptions : regroupe tous les services liés aux interruptions ou aux appels système. Ces services répondent aux interruptions matérielles entrantes en appelant les routines de gestion adéquates.
- ✓ Synchronisation : regroupe tous les services liés aux mécanismes de synchronisation internes au système d'exploitation.

4.3. Application

La dernière couche est la couche de l'application logicielle. La couche application est basée essentiellement sur la description de tâches applicatives. C'est la couche utilisée par le concepteur de logiciel pour représenter son application sans se préoccuper de l'architecture matérielle. Elle communique par API, la couche logicielle de plus bas niveau qui représente le système d'exploitation. Ainsi, à ce niveau de description, n'existent que des informations liées aux traitements à réaliser.

5. Architectures commerciales des systèmes monopuce multiprocesseur.

Dans cette partie, nous présenterons quelques exemples des architectures des systèmes monopuce multiprocesseur issues de l'industrie. Il existe un grand nombre des architectures, chacune spécifique à une application ou une domaine d'application. Nous avons choisi de présenter seulement les architectures qui ont un domaine d'application proche de l'application que nous avons développée pour notre expérimentation. Le but de cette présentation est d'analyser ces systèmes monopuce multiprocesseur qui semblent été réalisés par l'assemblage des composants matériels et dont les sous-systèmes sont conçus autour d'un processeur.

5.1. L'architecture Nexperia de Philips

La plateforme visuelle de Philips Nexperia [DUT01] est une architecture pour les applications visuelles numériques de divertissement. Cette plateforme est conçue pour la télévision

numérique, gestion de réseau à la maison, et applications de divertissement. Le premier membre de cette famille était la Viper PNX8500.

L'architecture de cette plateforme est montrée sur la figure 2.12. Elle est conçue autour d'un processeur VLIW Trimedia TM32 de Philips et un processeur MIPS PR3940 de 32 bits. Les deux processeurs sont interconnectés par trois bus de communication principaux : un pour le système de mémoire, et un pour le MIPS et un pour le processeur Trimedia. Il inclut également des ponts (bridges) pour connecter les trois bus entre eux. Le PNX8500 contient plusieurs accélérateurs : un moteur 2-D pour l'interprétation (rendering), un décodeur de vidéo MPEG2, une unité de traitement pour la composition d'image (image composition processor), une unité de traitement de flux vidéo d'entrée (video input processor), un processeur scalaire, et un processeur de système. Il a un contrôleur de mémoire pour l'interface externe de DRAM aussi bien qu'un contrôleur d'accès mémoire DMA pour chaque processeur. Il inclut également une variété d'entrées/sorties, incluant de UART, de PCI, d'IEEE 1394, et d'entrée audio numérique SPDIF.

Le processeur MIPS est utilisé pour exécuter le système d'exploitation et les tâches de contrôle. Il est possible d'exécuter différents systèmes d'exploitation, tels que le Windows CE, le WindRiver, le VxWorks, et le Linux. Le processeur partage quelques périphériques et utilise des sémaphores pour négocier la propriété de ces ressources partagées.

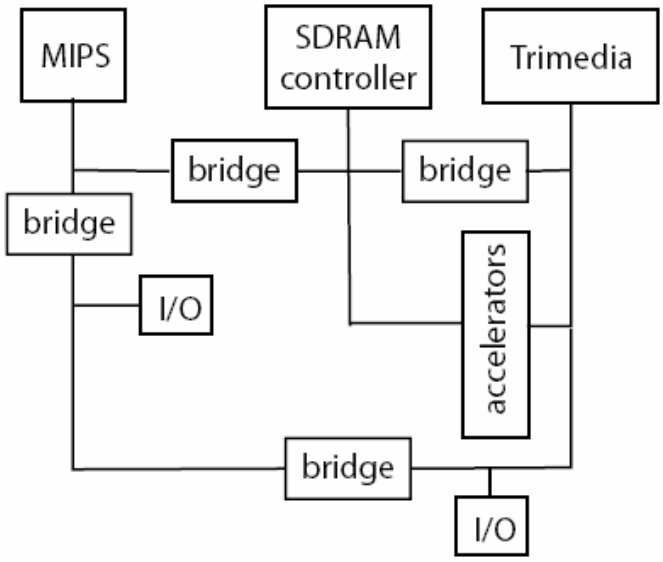


Figure 2.12 : L'architecture du système monopuce Nexperia PNX8500 de Philips.

5.2. L'architecture OMAP de Texas Instruments

L'architecture du TI OMAP™ [HEL02] est conçue pour des applications de la téléphonie 2.5G et 3G. En plus des services de voix de base, dans cette architecture sont prévues des possibilités pour le traitement de voix (speech processing), des services pour positionnement, sécurité, jeu, et multimédia.

La figure 2.13 montre l'architecture fonctionnelle du TI OMAP 5910. Elle contient un processeur spécifique TI C55x DSP et un processeur d'ARM9TDMI. Les deux processeurs partagent une interface externe de mémoire DRAM. La mémoire disponible pour le circuit est 192 Kbytes de SRAM partagée. Les périphériques intégrés sont une interface USB, une interface I^2C, des UART, des GPIO, et quatre interfaces spécifiques.

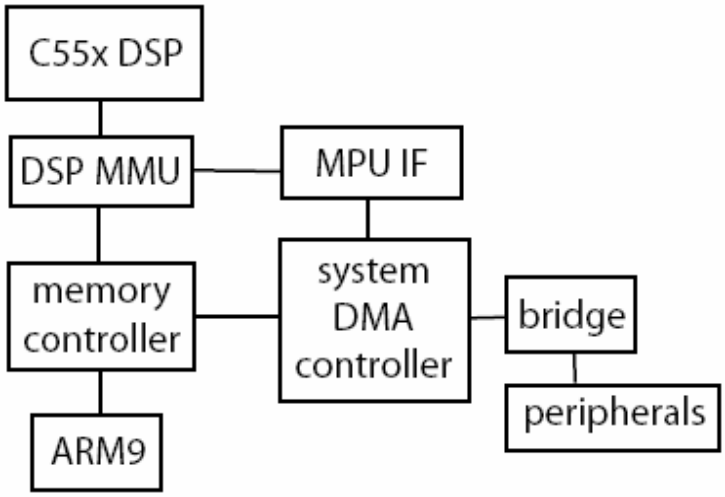


Figure 2.13 : L'architecture du système monopuce OMAP5910 de TI.

Le logiciel de ressource de DSP tourne sur le processeur ARM pour contrôler les tâches sur le DSP. Le DSP possède son propre OS en temps réel pour implémenter les tâches de contrôle de DSP.

5.3. L'architecture Nomadik de STMicroelectronics

L'architecture de ST Nomadik [ART03] est conçue pour des applications multimédia portables. Ceci exige le support de codage et de décodage de MPEG4. Il exige également un soutien d'une vaste gamme des tailles d'affichage, pour des téléphones portables (160 x 160) à PDA (320 x 240 jusqu'à 680 x 480).

La figure 2.14 montre la première implémentation de l'architecture hétérogène du système monopuce multiprocesseur de ST Nomadik. L'unité centrale de traitement principale pour le système est un ARM926E-JS. Ce processeur est utilisé pour le contrôle et la coordination des tâches, tandis que la plupart des fonctions sont exécutées par les accélérateurs spécifiques à l'application. La première version de Nomadik inclut un accélérateur audio et un accélérateur de vidéo. Tout ce matériel est relié autour d'un bus AMBA.

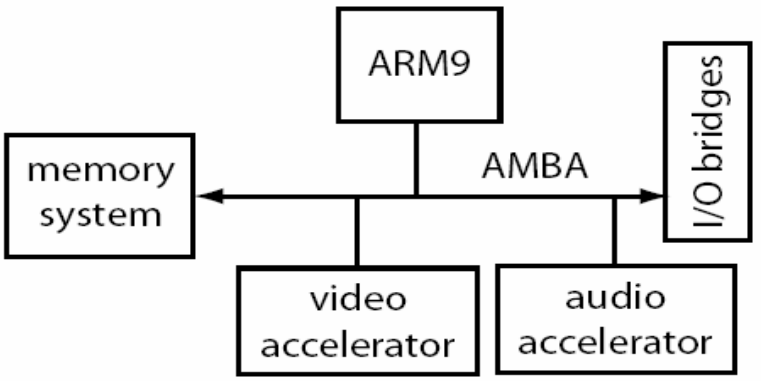


Figure 2.14 : L'architecture du système monopuce Nomadik de STMicroelectronics.

Chacun des accélérateurs audio et vidéo est conçu autour d'un processeur spécifique MMDSP+ de STMicroelectronics, qui fournit l'arithmétique de nombre entier de 16 ou de 24-bits et l'arithmétique de 32 bits en virgule flottante. L'accélérateur de vidéo inclut un MMDSP+, une unité de compression d'entrée d'image, l'unité de codage de vidéo, l'unité de traitement d'image (picture processing unit), et un contrôleur d'interruption. L'accélérateur de vidéo inclut deux bus : un qui relie le MMDSP+, les unités de matériel, la mémoire, et les contrôleurs d'interruption et l'autre qui relie les unités de matériel entre eux et au bus AMBA. L'accélérateur audio se base moins sur des unités de matériel. Il inclut un MMDSP+, la mémoire, des unités d'I/O, et deux bus internes.

L'architecture de Nomadik emploie des techniques à tous les niveaux pour réduire au minimum la consommation : les processeurs hétérogènes distribués, les algorithmes efficaces pour l'évaluation de mouvement et d'autres étapes intensives (consommant) du calcul, la compression de données sur le bus, la mémoire embarquée, les jeux d'instruction spécialisées, et la gestion dynamique de puissance. Nomadik consomme 20 mW tout en décodant 15 images par seconde avec la résolution QCIF en utilisant le standard MPEG4, y compris l'acoustique MP3.

Le noyau d'ARM de Nomadik permet au logiciel d'être mis en communication facilement à la plateforme. L'architecture Nomadik est compatible avec l'architecture d'OMAP de Texas Instruments.

II. Les niveaux d'abstractions, les outils et les langages de conception

Dans cette deuxième section, nous présenterons les différents niveaux d'abstraction, les langages et les outils utilisés pour la conception des systèmes monopuce.

1. Les différentes niveaux d'abstraction correspondant aux étapes de conception

Les systèmes sur puce sont un assemblage complexe des composants hétérogènes. Afin de simplifier le processus de conception, les concepteurs emploient généralement un certain nombre de modèles ou descriptions intermédiaires. Les modèles intermédiaires découpent la conception entière en plusieurs plus petites étapes de conception, dont chacune à un objectif de conception spécifique. Puisque les modèles peuvent être simulés et estimés, le résultat de chacune de ces étapes de conception peut indépendamment être validé.

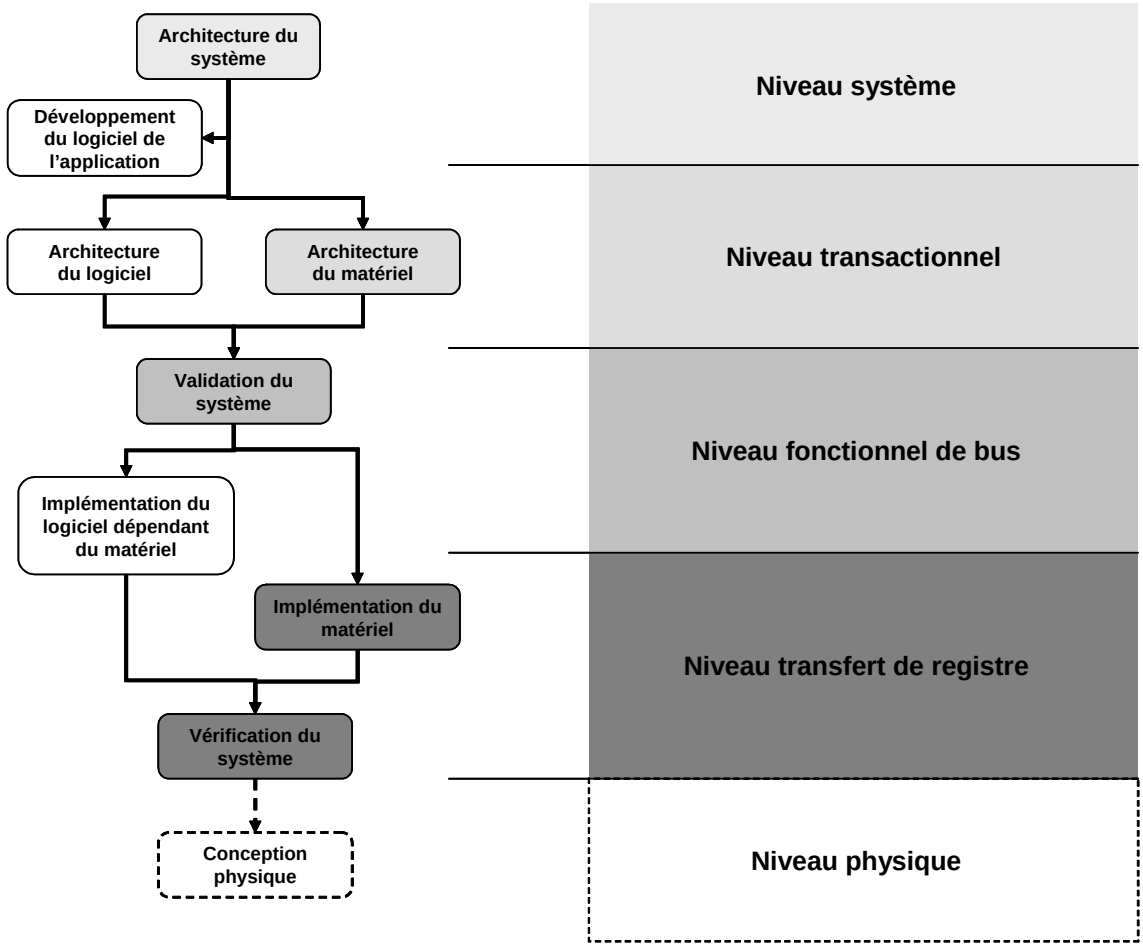


Figure 2.15 : Les différents niveaux d'abstraction et les étapes de passage d'un niveau à l'autre.

Modéliser le prototype d'un système aux différents niveaux est nécessaire, pour deux raisons : (1) fournir un niveau de conception correct pour valider les tâches de la spécification par rapport à un modèle d'un système, et (2) fournir une vitesse suffisante de simulation pour la tâche exécutée. Il y a quatre tâches de base qui doivent être soutenues par des modèles de prototype d'un système : (1) définition de l'architecture globale et développement de logiciel, (2) exploration de l'architecture et développement de logiciel personnalisé pour l'architecture, (3) validation de

l'architecture finale du système, et (4) l'implémentation des composants et intégration/vérification du système.

Ainsi on peut définir généralement cinq différents niveaux d'abstraction : le niveau système, le niveau transactionnel, le niveau de bus fonctionnel, le niveau transfert de registres et le niveau physique pour concevoir le matériel dans les systèmes sur puce. Sur la figure 2.15 chaque niveau d'abstraction de la conception d'un système est supposé contre les étapes de passage d'un niveau d'abstraction à un autre. L'ensemble des niveaux d'abstractions de système, que nous employons, sont similaires avec ceux décrits dans la documentation OSCI [HAV02] et d'autres publications [LEN00] et [NIC02a].

1.1. Le niveau système

La conception d'un système monopuce commence toujours par la spécification. La spécification d'un système est une description qui contient toutes les fonctionnalités et conditions que doit remplir l'application. Elle est indépendante de tous les détails d'implémentation du système. Cette description informelle n'est pas décrite avec un langage de conception fixe et elle n'est pas exploitable par des outils de conception.

L'objectif du concepteur de système est de créer un modèle formel du système, constitue d'un ou plusieurs sous-systèmes fonctionnels, basé sur ce modèle informel de la spécification. Chaque sous-système peut avoir plusieurs tâches qui modélisent la fonctionnalité de l'ensemble du système. La communication entre les sous-systèmes ou des tâches sont réalisées par des primitives d'un langage de haut niveau fournis par l'environnement d'exécution. Par exemple, la communication peut être modélisée par l'accès aux variables sans employer le concept de canal de communication.

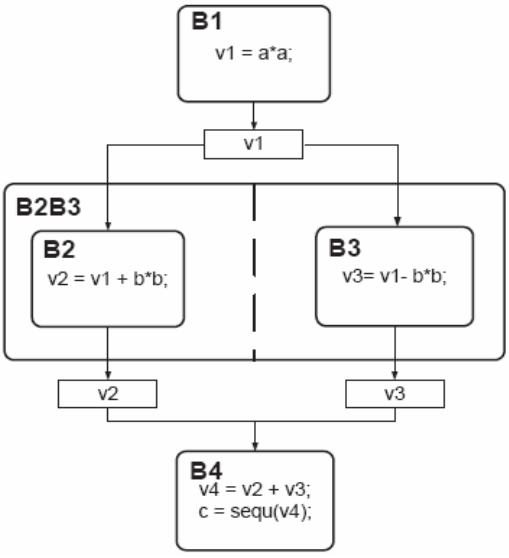


Figure 2.16 : L'exemple d'un modèle fonctionnel.

La figure 2.16 montre un exemple d'un modèle de spécification qui était présenté par Gajski [CAI03]. Les processus B1, B2, B3, et B4 s'exécutent séquentiellement. B2B3 est une composition parallèle de B2 et de B3. Des variables v1, v2 et v3 sont employées pour transférer des données entre les processus.

L'étape de raffinement à ce niveau est le processus de l'exploration de l'architecture. Il consiste en un partitionnement de l'algorithme de l'application en fonctions qui seront réalisées en matériel et en fonctions qui seront implémentées dans le programme logiciel. Cette étape est utilisée pour explorer des différentes solutions architecturales afin de fixer un partitionnement de logiciel et de matériel optimal d'un algorithme.

1.2. Le niveau transactionnel ou l'architecture virtuelle.

Ce niveau d'abstraction décrit l'architecture du système mais en ne se préoccupant que des objectifs fonctionnels pour l'application du système, c'est-à-dire en excluant complètement tous les détails liés à la réalisation. Le système est réalisé sous la forme d'une architecture virtuelle composée par des composants virtuels.

Un composant virtuel peut implémenter un ensemble de tâches logicielles ou une fonction matérielle mais sans aucune caractéristique précise pour le type du composant ou de sa structure interne. Le composant virtuel peut être réalisé comme un matériel câblé, ou un processeur d'usage universel, ou un DSP, en exécutant une fonction logicielle.

La description du système au niveau architecture virtuelle est un ensemble de tels composants virtuels travaillant concurremment qui communiquent par des canaux de communications abstraites. Ces canaux ne seront pas encore détaillés (appels de processeur externe, pour des liaisons avec des mécanismes des mémoires partagés).

Les canaux de communication utilisent des primitives transactionnelles définit par la norme TLM (Transaction Level Modeling) [HAV02], pour représenter seulement le transfert ou le processus de synchronisation de données entre les composants virtuels sans aucune information sur l'implémentation du protocole de communication. C'est-à-dire, une demande d'une donnée ou d'un bloc des données est accomplie dans une seule transaction, et le temps est indiqué comme « temps passé » plutôt qu'un événement par horloge. La communication à ce niveau d'abstraction est non temporelle (untimed), alors que la partie de calcul est précise approximativement au temps (approximatly timed) en estimant le temps d'exécution sur les composants spécifiques.

Le modèle du système à ce niveau est toutefois exploitable en utilisant des outils et des méthodes pour l'analyse de performances. Ils explorent les différentes solutions possibles pour le partitionnement des tâches afin de définir une architecture optimale du système.

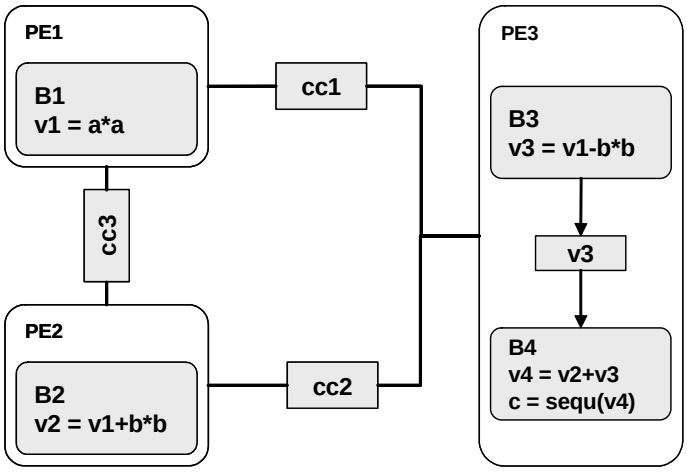


Figure 2.17 : L'exemple d'une architecture virtuelle.

Dans la comparaison au modèle de spécification, le modèle de l'architecture virtuelle indique explicitement comment chaque tâche est assignée à un composant de logiciel ou de matériel dans l'architecture du système. Un exemple d'une telle architecture est montré sur la figure 2.17. PE1, PE2 et PE3 sont des composants virtuels assignés aux processus B1, B2, B3, et B4 et cc1, cc2, cc3 sont les canaux de communication.

La synthèse de niveau système est l'étape de raffinement de ce niveau. La synthèse de niveau système représente une phase de génération automatique des composants de matériel à partir d'une bibliothèque extensible des composants de base réutilisables, développées pour un ensemble donné de protocoles et de composants, alors que la « synthèse » de la partie logicielle comporte la génération automatique d'un système d'exploitation spécifique à l'application. Cette étape de raffinement transforme le modèle virtuel d'architecture, contenant les modules abstraits de logiciel et de matériel, en une architecture contenant plusieurs sous-systèmes associés aux composants de logiciel et de matériel choisis par rapport aux besoins des services nécessaires. La génération de logiciel produit également un OS optimisé pour chaque type de processeur dans le système. Plus de détails au sujet du raffinement de matériel et logiciel employant l'environnement de ROSES peuvent être trouvés dans [CES02].

1.3. Le niveau de bus fonctionnel ou le prototype virtuel.

Les systèmes décrits à ce niveau possèdent des informations sur l'implémentation des protocoles de communication et des composants de matériel et de logiciel utilisés dans les sous-systèmes de l'application.

Au niveau de bus fonctionnel la communication est précise au niveau cycle, tandis que les composants de matériel reste fonctionnels et approximatifs par rapport au temps d'exécution. Dans le modèle fonctionnel de bus, les canaux virtuels transactionnels de niveau précédent sont

remplacés par des canaux de transfert. Un canal de transfert est précis au niveau cycle avec des signaux représentés par instanciation d'une variable. Des données sont transférées d'après l'ordre précis d'un protocole de bus. L'abstraction vient essentiellement des liens qui permettent de décrire un ensemble des signaux physiques par un seul canal logique. Les primitives utilisées sont usuellement « write(addr, data, ctrl) » et « read(addr, data, ctrl) ». L'adressage est explicite afin de designer l'élément correspondant concerné qui est en fait une mémoire ou un registre.

La partie logicielle à ce niveau est représentée par un programme et un système d'exploitation. En cours de conception le logiciel peut donc être simulé par un simulateur de processeur. Tout le logiciel à ce niveau est compilé et assemblé avec le jeu d'instruction du processeur cible.

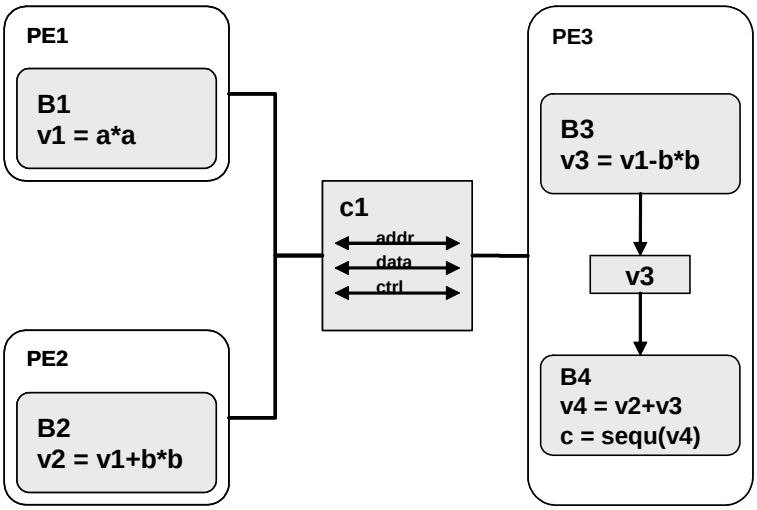


Figure 2.18 : L'exemple d'un modèle au niveau de bus fonctionnel.

La validation du modèle de système à ce niveau est réalisée par la co-simulation de matériel et de logiciel. Le modèle de simulation du système à ce niveau se compose :

- ✓ des sous-systèmes logiciels : représentés par des simulateurs du processeur au jeu d'instruction et du programme en langage machine contenant le logiciel de l'application. Il faut noter qu'au niveau de bus fonctionnel le programme en langage machine est un fichier qui est interprété directement par le simulateur de processeur. Tandis que au niveau plus bas comme RTL, le même programme en langage machine est un fichier compris par le simulateur de matériel comme le contenu d'une mémoire.
- ✓ des sous-systèmes matériels : représentés par des composants comportementaux utilisant des modèles fonctionnels de bus,
- ✓ et d'un bus de cosimulation : On utilise un bus de cosimulation car plusieurs simulateurs sont utilisés pour valider le système. Le bus de co-simulation est un adaptateur de simulateurs qui permet la communication entre les différents simulateurs. Il repose sur les mécanismes propres au système d'exploitation sur lequel s'exécutent les

simulateurs (par exemple pipeUNIX). Il existe des nouveaux outils de conception qui proposent un simulateur universel donnant la possibilité de simuler des composants de natures différentes dans un même environnement. Plus de détails sur le modèle de simulation et les outils de simulation seront donnés dans le chapitre suivant.

L'étape de raffinement du système vers le niveau RTL, consiste en l'intégration efficace des sous-systèmes de logiciel avec des composants de matériel existants. Les difficultés principales dans ce processus sont les différences entre les deux modèles. Le modèle fonctionnel de bus ne tient pas compte de la logique additionnelle requise pour réaliser la communication entre le processeur et son environnement ou du code supplémentaire nécessaire pour certaines initialisations du processeur caché par le débuggeur du simulateur de processeur.

Nous reviendrons sur ces difficultés pendant le processus de prototypage d'un système sur puce multiprocesseur dans le chapitre suivant en proposant un flot d'intégration des systèmes monopuce du niveau de bus fonctionnel au niveau de transfert de registre.

1.4. Le niveau RTL

Le niveau RTL est clairement identifié. A ce niveau de conception, la communication interne et externe des composants matériels est réalisée par des fils et des bus physiques. Les échanges internes dans ces composants matériels sont connues au cycle d'horloge. La granularité de l'unité de temps est le cycle horloge et les primitives de communication sont des lectures et écritures sur des ports et l'attente d'un nouveau cycle d'horloge. La description est dite précise au niveau des portes ou « pin accurate ».

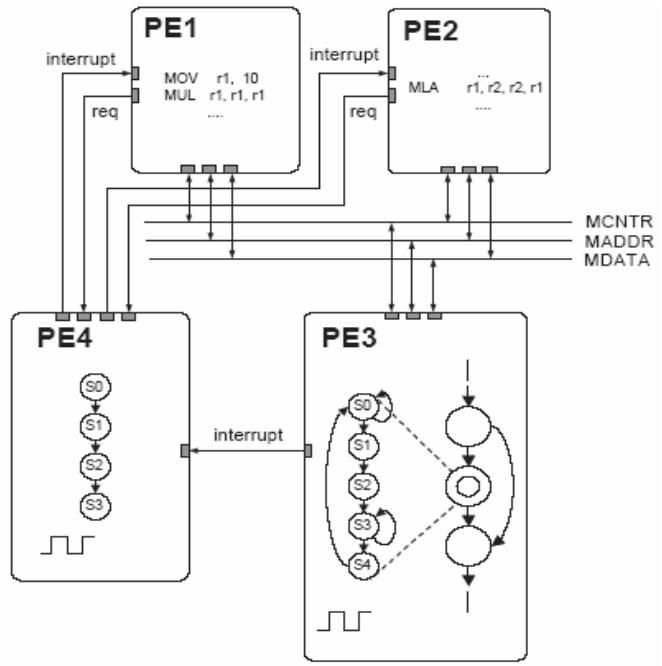


Figure 2.19 : L'exemple d'un modèle au niveau RTL.

La figure 2.19 montre un exemple du modèle au niveau RTL. PE1 et PE2 sont des microprocesseurs tandis que PE3 et PE4 sont implémentés comme des composants de matériels câblés.

Il permet de produire tous les niveaux plus détaillés par génération automatique ou par synthèse. On peut aussi tout simplement en disposer par réutilisation des composants déjà implémentés. Ce niveau est de plus en plus considéré comme trop détaillé en approche système. D'une part il nécessite un travail important pour le décrire complètement, d'autre part son utilisation pour des vérifications par simulation conduit à des temps excessifs.

1.5. Le niveau physique

Le niveau physique englobe toutes les étapes de conception permettent d'élaborer le layout d'un circuit correspondant à la description au niveau transfert de registre. En effet le nombre des étapes de la conception physique a augmenté avec le développement des technologies submicroniques profondes.

En passant par les étapes de validation et d'implémentation physique on arrive au niveau logique qui est un ensemble de portes logiques simples et des bascules, implémentant les équations booléennes. Ensuite le niveau circuit représente le système comme un schéma en transistors qui comporte aussi les caractéristiques physiques de chaque transistor (type, taille, paramètres). Pour arriver au niveau le plus bas qui représente le dessin des masques du circuit. Il s'agit d'une liste de polygones représentant les différentes couches physiques du circuit (diffusion, polysilicium, métal1, metal2, etc.).

2. L'état de l'art sur les langages utilisés pour la conception des systèmes embarqués

Le langage de conception idéal doit couvrir tous les niveaux d'abstraction, tout en permettant les différents types d'utilisation, documentation, vérification, ou simulation et raffinement. Malheureusement, ce langage n'existe pas encore. Cette section présente les langages couramment utilisés pour la conception des systèmes électroniques.

2.1. Langages de description matérielle proprement dit, HDL (Hardware Description Languages)

Les langages de description matérielle sont des langages qui supportent les concepts spécifiques aux systèmes matériels tels que les concepts de temps, de parallélisme, de réactivité et

de communication interprocessus (signaux et protocoles). Les deux langages de description matérielle les plus connus sont VHDL et Verilog.

2.1.1. Langage VHDL

Le langage VHDL est un langage de description du matériel, destiné à décrire le comportement et/ou l'architecture d'un module matériel, c'est-à-dire une fonction combinatoire et/ou séquentielle. La version initiale de VHDL est un standard de IEEE 1076-1987 [IEE88]. Elle permet la description de tous les aspects d'un système matériel : son comportement, sa structure et ses caractéristiques temporelles. Par système matériel, on entend un système électronique arbitrairement complexe réalisé sous la forme d'un circuit intégré ou d'un ensemble de cartes. Le comportement définit la ou les fonctions que le système remplit (p. ex. le comportement d'un microprocesseur comporte, entre autres, des fonctions arithmétiques et logiques). La structure définit l'organisation du système en une hiérarchie de composants (p. ex. un microprocesseur est constitué d'une unité de contrôle et d'une unité opérative; cette dernière est elle-même, entre autres, constituée d'un composant réalisant les opérations arithmétiques entières et d'un composant réalisant les opérations arithmétiques en virgule flottante). Les caractéristiques temporelles définissent des contraintes sur le comportement du système (p. ex. les signaux d'un bus de données doivent être stables depuis un temps minimum donné par rapport à un flanc d'horloge pour qu'une opération d'écriture dans la mémoire soit valable).

La description d'un système matériel en VHDL est apte à être simulée, ce qu'on appelle un *modèle exécutable* (*executable model*). Il est possible de lui appliquer des *vecteurs de test (testbench)*, également décrits en VHDL et d'observer l'évolution des signaux du modèle dans le temps. La sémantique d'une description VHDL est basée sur le principe de la *simulation discrète dirigée par les événements* (*event-driven discrete simulation*). La description VHDL ne peut prendre un nouvel état en simulation que lorsqu'un stimulus change de valeur et que ce changement est propagé dans la description. Le langage VHDL définit des règles précises pour l'évaluation de l'état d'une description en présence d'un changement de stimuli. Ces règles garantissent que l'évaluation abouti au même résultat quel que soit l'outil de simulation utilisé.

Le langage VHDL est aussi utilisé pour la synthèse, par exemple pour dériver automatiquement un circuit à base de portes logiques optimisé à partir d'une description au niveau RTL ou algorithmique. Cette application très importante du langage sort toutefois du cadre de sa définition standard et comporte des limitations dont les plus importantes seront présentées au chapitre 3.

2.1.2. Langage Verilog

Le langage de description de matériel Verilog, est un autre standard IEEE 1364-1995 [IEE96] qui a suivi un développement différent du langage VHDL.

On peut noter que Verilog est plus spécialement conçu pour modéliser des circuits intégrés logiques (jusqu'au niveau du transistor), alors que VHDL permet la modélisation de systèmes plus abstraits. Verilog est aussi un langage moins verbeux que VHDL et qui hérite beaucoup du langage de programmation C. Il offre moins de possibilités d'extensions (nombre limité de types prédéfinis, par exemple) et offre moins de possibilités de gestion des modèles que VHDL. Bien que Verilog 2001 [SUT02] lève en partie ces limitations (bloc de génération par exemple), il reste que VHDL et Verilog restent deux approches à la fois partiellement recouvrantes et complémentaires. Il existe d'ailleurs sur le marché des outils capables de simuler des modèles mixtes VHDL et Verilog. Ces deux langages s'influencent d'autre part l'un à l'autre : Verilog 2001 inclut des mécanismes VHDL et VHDL pourrait bien faire de même avec des mécanismes Verilog dans une prochaine révision du standard.

2.2. Langages de description de matériel basés sur des langages de programmation.

Certaines approches ont étendu les langages utilisés en informatique pour la programmation (C, C++) en y ajoutant les concepts spécifiques aux matériels, présentés ci-dessus. Le choix de ces langages se base sur trois raisons principales : ils fournissent le contrôle et le type de données nécessaires, la plupart des systèmes contient des parties matérielles et logicielles et donc l'un de ces langages représente un choix naturel pour la partie logicielle. De plus les concepteurs sont familiers avec ces langages et les outils conçus autour. Les langages développés consistent en la définition de fonctions ou de classes permettant de modéliser la concurrence, le temps et le comportement réactif. Les langages de ce type les plus utilisés sont SystemC [SYS03] et SpecC [GAJ00].

2.2.1. Langage SpecC

SpecC [GAJ00] : langage de description et de spécification système basé sur le C, avec des extensions syntaxiques (qui le rendent incompatible à la norme ANSI). Ces extensions incluent, par exemple, des types de données adéquats, des commandes pour supporter la concurrence et la description de machines à états. Il a été développé à l'université de Californie, Irvine. Au delà du langage lui même, les travaux menés par l'équipe de D. Gajski introduisent des concepts

intéressants pour la modélisation système et les techniques de raffinement. Le développement de SpecC se fait dans le cadre du STOC (SpecC Technology Open Consortium).

2.2.2. Langage SystemC

SystemC [SYS03] est aussi un langage de description de matériel qui a pour objectif de modéliser des systèmes numériques *matériels et logiciels* à l'aide de C++. Il modélise donc non seulement des systèmes *matériels*, mais aussi des systèmes *logiciels*, *mixtes* ou *non partitionnés*.

SystemC est une « extension » du langage C++, plus précisément *une bibliothèque de classes C++* contenant des modèles de matériel, ainsi que toutes les primitives de base pour modéliser un système matériel qui n'aurait pas été incluses dans la bibliothèque de base.

SystemC introduit de nouveaux concepts (par rapport au C++ classique) afin de supporter la modélisation et la description du hardware et ses caractéristiques inhérentes comme la concurrence et l'aspect temporel. Ces nouveaux concepts sont implémentés par des classes C++ tels que les modules, ports, interfaces, canaux, processus (threads et methods).

D'autre part, seulement un sous ensemble du langage est synthétisable. Il est équivalent à celui d'un langage HDL classique (Verilog ou VHDL), et les outils actuels utilisent d'ailleurs le même noyau de synthèse pour synthétiser les descriptions. Ce sous ensemble n'est certainement pas figé et pourrait varier en fonction des outils futurs.

En résumé SystemC est une bibliothèque C++, qui fournit des classes C++ adaptées à la modélisation de matériel et du logiciel ainsi qu'un moteur de simulation événementiel rapide, le tout sous licence libre. Il permet de garder un même langage d'un bout à l'autre du flot de conception, excepté peut-être à la toute fin (mais on a alors des moyens de s'assurer de la cohérence des modèles).

2.2.3. Langage SystemVerilog

SystemVerilog [VER03] est la dernière norme récemment ratifiée pour un langage de description et vérification de matériel (HDL). SystemVerilog est une prolongation importante du langage Verilog, qui était développée par la société « Accellera » pour améliorer la productivité dans la conception des systèmes sur puce. SystemVerilog est visé principalement pour des flots de conception et vérification physique des circuits intégrés avec des liens puissants pour le flot de conception au niveau système. Son amélioration la plus importante par rapport au langage Verilog est le soutien pour la modélisation et la vérification des composants transactionnel ce qui permet d'appeler directement des fonctions de C/C++/SystemC, et vice versa. SystemVerilog permet donc la co-simulation efficace avec des blocs décrits en SystemC – un apport important entre la conception au niveau système et l'implémentation et la vérification des circuits intégrés.

3. Les outils de conception

Cette section présentera les outils aidant les concepteurs lors de développement d'un système. Elle n'a pas pour but de présenter de manière exhaustive l'ensemble des outils existants actuellement. C'est pourquoi on discutera que les outils qui ont fait partie d'une ou autre façon dans le processus du travail de cette thèse.

3.1. L'environnement ROSES de TIMA-SLS

L'environnement de conception ROSES [CES02] du groupe SLS permet *la génération automatique d'interfaces logiciel/matériel* des systèmes monopuce. L'interface logiciel/matériel générée est un adaptateur de communication entre les tâches exécutées par le processeur et les ressources matérielles. Plus précisément l'interface logicielle permet au code de l'application d'accéder aux ressources matérielles du processeur et l'interface matérielle permet au processeur d'accéder au réseau de communication

Les interfaces matérielles prise en compte par ROSES concernant les processeurs [LYO03], mais aussi la mémoire [GHA03] et les composants matériels spécifiques [GRA04]. Les interfaces logicielles sont le système d'exploitation [GAU02] et les couches de communication [PAV04]. L'intérêt de ROSES est la génération automatique des interfaces à partir d'une spécification du système et de quelques caractéristiques pour guider la conception (protocoles, adressage, et autres paramètres) De plus ce flot permet une validation à plusieurs niveaux d'abstraction à l'aide de mécanismes de co-simulation [NIC02].

Le principe le plus important dans ce flot est la conception d'un système complet par assemblage d'éléments [CES02]. Que ce soit pour composer les parties logicielles des interfaces ou les parties matérielles mais aussi le développement des modèles de simulation, la technique reste la même: elle consiste en un assemblage d'éléments de bibliothèque. Ces bibliothèques doivent contenir des composants de base à partir desquels le système sera assemblé. Ces composants de base doivent être configurables et ils peuvent être conçus pour différents standards.

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

- ✓ ASAG (Application Specific Architecture Generator) est l'outil de génération des interfaces matérielles.
- ✓ ASOG (Application Specific Operating System Generator) est l'outil de génération des interfaces logicielles.
- ✓ CosimX est l'outil de génération des interfaces pour la simulation.

La figure 2.20 résume le flot qui permet la génération d'interfaces à partir d'un système décrit au niveau architectural vers une description au niveau RTL. L'entrée du flot est décrite en langage de spécification développé par le groupe SLS. Ce langage est une extension de SystemC nommé VADeL (Virtual Architecture Description Language).

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

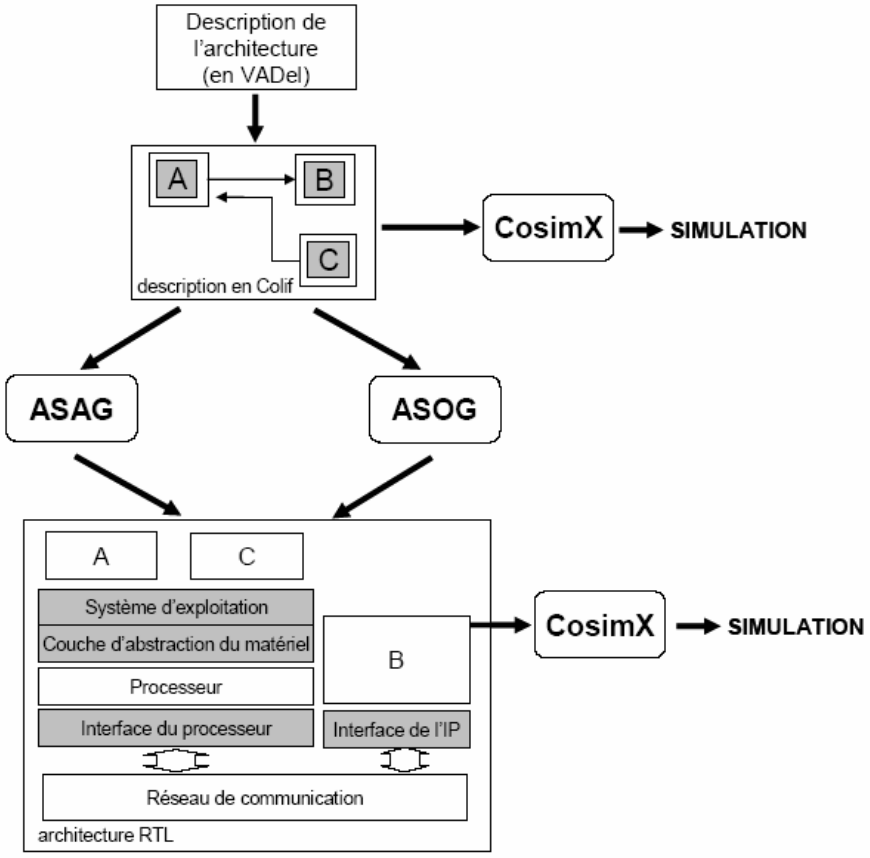


Figure 2.20 : Le flot de conception ROSES développé par le groupe SLS du laboratoire TIMA.

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

L'outil ASOG a été développé par Lovic Gauthier [GAU02]. Cet outil génère l'interface logicielle qui consiste en un système d'exploitation spécifique et une couche d'abstraction du

matériel à partir de paramètres donnés dans les ports en COLIF. Ces paramètres décrivent les services de communication demandés et requis par les tâches.

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

CosimX développé par Gabrielle Nicolescu [NIC02] puis par Adriano Sarmento est utilisé pour valider les systèmes décrits à plusieurs niveaux d'abstraction. Il permet de rendre exécutable la description VADeL du système. L'exécution du système avant génération des interfaces permet de valider le comportement des tâches, des composants matériels ainsi que le choix des services de communication. CosimX offre aussi la possibilité de tester le fonctionnement du système par cosimulation après le raffinement partiel ou complet du système.

Une méthodologie de conception de haut en bas basée sur la synthèse des interfaces permet l'exploration rapide des solutions pour le partitionnement de matériel et de logiciel et de mapper des spécifications exécutables en SystemC avec des fonctions ou des blocs réutilisables.

En utilisant ce flot de conception, le concepteur peut rapidement évaluer l'implémentation optimale des fonctions sur un modèle représentatif de matériel/logiciel qui est assez précis pour analyser tous les effets du bus et de l'architecture de mémoire.

3.2. L'outil ConvergenSC de CoWare et l'outil Incisive de Cadence

Les sociétés CoWare et Cadence proposent un tel flot de conception unifié entre les outils de CoWare de la famille ConvergenSC et la plateforme de vérification de Cadence Incisive. La combinaison de ConvergenSC et de la plateforme Incisive fournit une solution puissante et complète pour la conception et la vérification basée sur le langage de description de matériel SystemC.

3.2.1. L'outil « ConvergenSC » de la société CoWare

« ConvergenSC » [HAR03] de CoWare est un environnement de conception et de vérification basée sur le langage SystemC.

La famille des outils « ConvergenSC » est composé principalement de trois outils :

- ✓ L'outil « Platform Creator » donne un environnement graphique puissant pour un assemblage, une configuration et une optimisation rapide des composants au niveau transactionnel ou au niveau de bus fonctionnel décrit en SystemC.

- ✓ L'outil « System Verifier » est un simulateur HDL SystemC qui permet la vérification et la co-simulation de matériel/logiciel au niveau système.
- ✓ L'outil « System Designer » est un moyen très performant pour l'estimation de la performance, l'analyse et le déboguage des systèmes sur puce complexes.

L'environnement ConvergenSC possède la possibilité d'importation des composants matériels décrits au niveau RTL. Ce qui donne la possibilité de simuler des différentes parties d'un système en cours de développement aux différents niveaux d'abstractions. L'outil ConvergenSC automatise l'intégration des modules matériels décrits en VHDL ou en Verilog avec des modules matériels décrits en SystemC au niveau transactionnel, y compris la génération automatique des adaptateurs nécessaires pour interconnecter les interfaces des composants au niveau RTL avec les interfaces des composants au niveau transactionnel ou au niveau de bus fonctionnel.

3.2.2. L'outil « Incisive » de la société Cadence

Le simulateur universel « Incisive » de Cadence [CAD05], fait partie de la plateforme de vérification Incisive, qui fournit une vaste gamme des moyens pour la vérification des systèmes sur puce. Cette plateforme de vérification soutient tous les langages de conception matérielle comme Verilog, SystemVerilog, VHDL, SystemC, PSL (Property Specification Language) et OVL. Incisive représente un environnement de vérification comprenant un soutien complet pour la conception au niveau transactionnel et la génération des vecteurs de test unifiée.

L'objectif général du flot de conception entre les outils de ConvergenSC et la plateforme Incisive est d'effectuer plus de travail de conception et de vérification au niveau transactionnel. Parce que les modèles au niveau transactionnel peuvent être plus facilement développés et peuvent être simulés beaucoup plus rapidement, en permettant aux concepteurs d'effectuer une plus longue exploration au niveau système et une validation de la fonctionnalité plus tôt dans le flot de conception. En plus, les composants et les vecteurs de test conçus au niveau système peuvent être réutilisés pour vérifier les composant au niveau plus bas.

Ainsi, dans le flot de conception de CoWare et Cadence, tous les composants de système sont initialement représentés au niveau transactionnel, et aucun bloc n'est modélisé en utilisant un langage de description matériel au niveau RTL. Dans la pratique, cependant, il est possible que les composants soient disponibles au niveau RTL. Dans ce cas-ci, le flot de CoWare et Cadence offre des possibilités d'employer ces modèles également dans la phase de conception au niveau système en utilisant une co-simulation multi-niveau et multi-langage.

3.3. L'outil « MaxSim » de la société ARM

MaxSim [LEN05] est un outil unifié pour la conception, la simulation et le déboguage des systèmes sur puce. Il utilise des modèles de composants précis au niveau du cycle décrits en langage SystemC, ce qui offre une exactitude et une performance de la simulation amélioré par rapport à des environnements de simulation classique de matériel. MaxSim possède aussi des possibilités de déboguage et de profilage qui sont très utiles pour l'exploration des architectures au niveau transactionnel et le développement du logiciel embarqué.

MaxSim donne aux concepteurs de système et de matériel la possibilité d'explorer rapidement et exactement des différentes architectures pour leur système. La technologie des prototypes virtuels employé dans MaxSim permet aux ingénieurs du logiciel embarqué de réduire le temps de conception en commençant par le développement et le codage du logiciel dépendant du matériel avant que les composants matériels soient disponibles. Les intégrateurs de matériel et de logiciel peuvent utiliser les possibilités de co-simulation de MaxSim pour identifier et résoudre les problèmes plus tôt dans la conception, en diminuant le risque d'apparition des problèmes plus tard dans la conception.

L'outil MaxSim utilise une bibliothèque des composants décrits en SystemC. Cette bibliothèque peut être étendue par les utilisateurs de MaxSim en utilisant un magicien « wizard » qui permet aux utilisateurs de rajouter facilement des nouvelles descriptions de composants en SystemC.

L'outil MaxSim sera détaillé en plus dans le chapitre suivant, où nous allons décrire l'utilisation de MaxSim pour le prototypage des systèmes monopuce.

3.4. L'outil System Studio de Synopsys

Le « System Studio » [SYN03] est un environnement de conception au niveau système. Il est utilisé principalement pour la conception des architectures et des algorithmes au niveau système des applications SoC innovatrices. La conception d'algorithmes couvre le traitement des signaux tel que la téléphonie, le codage de multimédia, le DSL et les modems sans fil. La conception d'architecture implique de choisir bien les processeurs, la logique spécifique, les bus, les mémoires et les périphériques afin de réussir une utilisation plus efficace du silicium. La conception et l'intégration des éléments de matériel et de logiciel au niveau système, qui sont adressées par l'outil System Studio, implique l'utilisation d'un niveau d'abstraction plus haut que RTL.

Le System Studio est un outil unifié qui adresse efficacement tous les aspects importants à la conception et à la vérification au niveau système suivants de SoC :

- ✓ Capture et gestion de conception : Dans le System Studio les concepteurs peuvent travailler avec les différents niveaux d'abstraction afin de capturer et vérifier leur système entier dans un environnement unifié. Il a un rédacteur textuel et graphique intégré qui permet aux utilisateurs de capturer l'architecture et le comportement algorithmique de système aux divers niveaux d'abstraction.
- ✓ Modéliser des algorithmes : Les algorithmes sont capturés, vérifiés et optimisés à l'aide des graphiques intuitifs de flux de données et des machines d'état fini (FSM).
- ✓ Modéliser des architectures : Le System Studio fournit un soutien complet du langage SystemC. Des modèles décrits en SystemC créés ailleurs peuvent être facilement importés, ainsi que des nouveaux modèles peuvent être créés rapidement en utilisant l'aide d'un magicien « wizard ».
- ✓ Simulation : L'outil possède un moteur de simulation, qui évalue automatiquement les différents modèles étant employés (flux de données, FSM ou SystemC) et optimise la simulation du système par différentes techniques. Le System Studio fonctionne sans problème avec la plupart des simulateurs HDL tels que Synopsys VCS, Cadence NC-Sim et Mentor ModelSim pour pouvoir co-simuler avec des blocs au niveau RTL décrits en VHDL ou Verilog.

3.5. L'outil « coreTools » de Synopsys

La famille « coreTools » de Synopsys [SYN04] est un ensemble complet d'outils de réutilisation des composants IP (Intellectual Property). Ces outils permettent aux concepteurs de réaliser des gains de productivité dans la conception et la vérification des systèmes sur puce. On réduit le temps de conception global et on peut éliminer le risque d'erreurs dans l'intégration de sous-systèmes des applications SoC, en utilisant un flot de conception et de vérification basé sur l'assemblage des composants IP.

La famille « coreTools » inclut les outils suivants :

coreBuilder : un outil de paquetage qui permet aux concepteurs de capturer la structure et la fonctionnalité d'un composant IP et de créer une vue graphique pour un outil de configuration. Il soutient toutes les différentes vues possibles des composants IP requis par la conception et la vérification. Ceci réduit le temps de conception des composants IP, améliore la qualité et donne la possibilité de réutilisées ces composants IP.

coreAssembler : un outil de génération des composants IP qui peut automatiquement produire des interconnexions au niveau RTL entre les composants IP , aussi bien que de les configurer pour un système.

coreConsultant : l'environnement graphique pour configurer et synthétiser les différents composants IP emballés avec l'outil coreBuilder.

Les outils de la famille « coreTools » soutient la norme de standardisation pour le développement et la vérification des composants IP – SPIRITv1.0.

3.6. L'outil « Platform Express » et l'outil Seamless de la société Mentor Graphics

La société Mentor Graphics fournit un environnement pour la co-simulation de matériel et de logiciel de haut niveau.

3.6.1. L'outil « Platform Express » de Mentor

L'outil « Platform Express » [MEN0] fait partie d'une famille des outils de conception et de vérification de la société Mentor Graphics. « Platform Express » est un outil de conception utilisé pour concevoir et vérifier des sous-systèmes des applications monopuce basés sur des composants IP. Il permet de choisir parmi un ensemble optimisé des sous-systèmes de base, et d'élargir leurs fonctionnalités en intégrant des composants IP supplémentaires. L'outil « Platform Express » intègre les différents composants IP en utilisant un bus de communication comme par exemple le bus AMBA. En fournissant des sous-systèmes configurables optimisés et préfabriqués, l'outil « Platform Express » permet aux ingénieurs d'obtenir un gain de productivité en réutilisant des composants matériels et de se concentrer en détail sur les problèmes de la conception.

3.6.2. L'outil « Seamless » de Mentor

Seamless [MEN03] est un outil pour la co-vérification de matériel et de logiciel et pour l'analyse de performances des systèmes monopuce. Il permet d'interconnecter plusieurs simulateurs de processeur (ISS), grâce à l'interface SPI. Cette possibilité nous permet de simuler un système, multiprocesseur hétérogène. Seamless est compatible avec plusieurs outils de vérification et description comme ModelSim VHDL, Verilog XL, VSC etc.

4. Conclusion

Ce chapitre a présenté les composants logiciels et matériels utilisés dans la conception des architectures SoC et les méthodes comme des niveaux d'abstraction, des langages et les outils de conception, utilisées pour assembler ces composants. Nous avons consacré aussi une section avec des exemples des architectures matérielles existantes.

L'apport de ce chapitre est sur la définition et précision des termes et des composants que j'ai utilisé pendant l'élaboration de la partie pratique de cette thèse, et que j'utiliserai ensuite pour présenter les résultats et les contributions de mes travaux.

Chapitre 3

VERS UN CONCEPT POUR L'INTEGRATION DE MATERIEL ET DE LOGICIEL A PARTIR DU NIVEAU DE BUS FONCTIONNEL VERS LE NIVEAU RTL

Ce chapitre présente une des contributions principales de ces travaux de thèse. Il définit une méthodologie d'intégration systématique pour la transformation de la partie matérielle et la partie logicielle à partir du modèle au niveau de bus fonctionnel vers le niveau RTL. L'objectif de cette contribution est de faciliter le passage entre les deux niveaux d'abstraction et de définir une base pour des outils de conception automatisant le flot de conception.

1. Introduction

L'objectif principal de toutes les méthodologies de conception contemporaines est de pouvoir faire au plus tôt la validation de la fonctionnalité et de l'architecture d'un système. Pour permettre cette validation, ces méthodologies exigent une description de simulation du système de haut niveau. Une telle description doit fournir un niveau d'abstraction idéal pour surmonter les restrictions de la conception traditionnelle permettant la conception concourante du matériel et du logiciel. Le concept de prototype virtuel représente exactement une telle description de simulation du système en développement. Cette description est employée par les concepteurs pour le développement concourant de l'architecture du matériel et du logiciel d'application, la validation du système entier et la détection des problèmes d'intégration.

Plusieurs méthodologies et outils, comme par exemple [GHO95], [PRO02], [HAR03], [SYN04] et d'autres, sont proposés pour la conception des prototypes virtuels et pour fournir des étapes bien définies permettant le passage à partir des descriptions abstraites de plus haut niveau vers des descriptions au niveau RTL d'une manière cohérente. Les approches existantes sont conçues en particulier pour des plateformes spécifiques ou pour un ensemble des composants prédéfinis [KEU00], [CES02]. Une approche d'intégration systématique plus générique, à partir du modèle de prototype virtuel jusqu'à un modèle de matériel au niveau RTL, est exigée.

Ce chapitre présente un concept systématique pour raffiner un prototype virtuel dans un modèle de niveau RTL. Le raffinement est réalisé par une étape d'intégration de SoC visant la transformation de la partie de matériel et de la partie de logiciel du système dans un modèle de niveau RTL entièrement détaillée qui peut alimenter les flots de conception physique existants.

Le reste de ce chapitre présentera dans la section 2 les modèles de composants matériels et logiciels utilisés au niveau de bus fonctionnel. La section 3 présente les méthodes et les techniques de la validation par cosimulation des systèmes monopuce au niveau fonctionnel du bus, ainsi que l'environnement « MaxSim » utilisé pour modéliser et simuler le prototype virtuel. La section 4 présentera en détail, le flot d'intégration défini et les techniques utilisées pour l'étape d'intégration de matériel et de logiciel.

2. La conception au niveau de bus fonctionnel

Cette section présentera les descriptions d'entrée et de sortie au niveau de bus fonctionnel. Nous allons donner aussi une représentation détaillée des composants matériels et logiciels composants la description d'entrée.

2.1. La description d'entrée au niveau de bus fonctionnel

Dans la conception au niveau de bus fonctionnel, le prototype virtuel est la description initiale représentant l'architecture et l'interaction entre le matériel et le logiciel. Cette description (voir la figure 3.1.a) se compose d'une partie logicielle contenant une application, un système d'exploitation et une couche d'abstraction de matériel générique, typiquement décrivant les accès processeur/mémoire et d'une partie matérielle représentant tous les composants de matériel spécifiques du système. Très souvent la partie logicielle est simulée en utilisant un simulateur du processeur, qui a pour rôle de remplacer une description complète du processeur cible pour accélérer la validation. La partie matérielle est simulée en utilisant des descriptions comportementales des composants du système. Tous les composants constituant la description du système en cours de conception sont interconnectés par des interfaces fonctionnelles, qui servent à établir la communication entre les deux parties du système. Ces interfaces peuvent être soit des interfaces fonctionnels de bus qui auront une implémentation dans le circuit final, car il s'agit d'une interface liée avec le protocole de communication, soit une interface de co-simulation qui est utilisée seulement pour interconnecter deux simulateurs. Les interfaces fonctionnelles seront détaillées un peu plus tard dans le manuscrit.

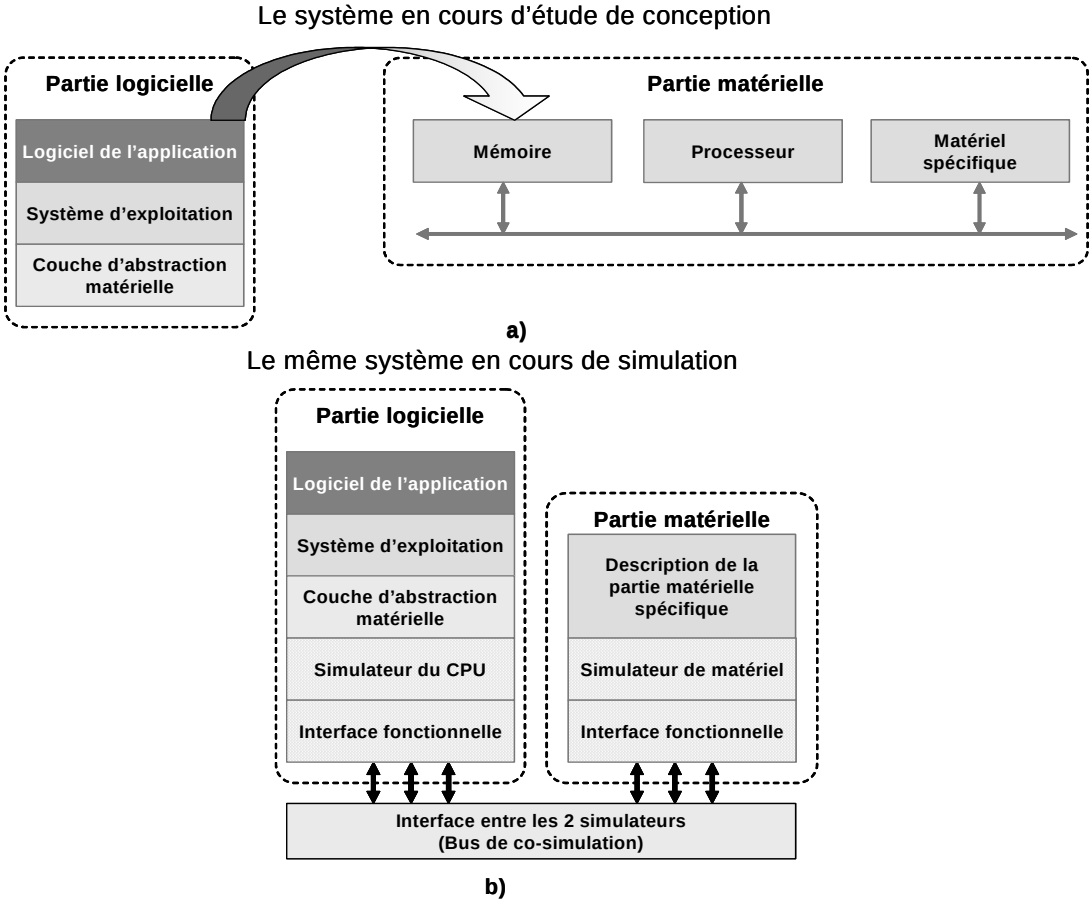


Figure 3.1 : La description du système en cours de conception au niveau de bus fonctionnel.

Normalement, le prototype virtuel d'un système au niveau de bus fonctionnel est assemblé à partir d'une bibliothèque en réutilisant des composants de propriété intellectuelle existants, pour accélérer le développement du système.

Même si ce modèle du système est très précis par rapport au nombre des composants implémentant l'architecture du système et son fonctionnement, il cache encore plusieurs détails qu'une représentation du système au niveau RTL décrit explicitement. Un simple exemple : au niveau de bus fonctionnel il n'est pas pertinent de savoir le mode d'accès à la mémoire c'est-à-dire si le processeur et la mémoire supportent ou non des accès mots par mots seulement ou « burst », ainsi que certains détails sur la réalisation de la communication et la synchronisation pour le matériel comme le décodage des adresses mémoire ou registres et la gestion des interruptions et l'arbitrage. De plus le simulateur de processeur ne contient pas les informations sur la mémoire contenant les programmes. Il n'accède qu'au fichier binaire exécutable.

2.2. Le concept de base pour la validation par cosimulation des systèmes monopuce au niveau de bus fonctionnel

Le principe de la cosimulation est l'exécution parallèle des simulateurs nécessaires pour la validation d'un système. Chaque simulateur exécute un module du système décrit dans un langage spécifique. Les modules ou sous-systèmes peuvent appartenir à différents modules de calcul. De plus, les modules peuvent être spécifiés aux différents niveaux d'abstraction car les passages d'un niveau à l'autre pour les différents modules ne sont pas simultanés. Il faut donc pouvoir travailler à plusieurs niveaux d'abstraction simultanément. Ainsi il est possible de simuler l'ensemble du système en coordonnant et en échangeant les données calculées et interprétés par chaque module. La cosimulation doit prendre en compte les problèmes de coordination et de synchronisation et doit permettre d'obtenir un résultat qui ne doit pas changer la fonctionnalité de la future réalisation.

2.3. Les composants matériels au niveau de bus fonctionnel

Pendant la conception d'un système monopuce, les ingénieurs utilisent souvent le processeur ou un autre composant matériel comme un générateur des transferts pour l'étape de vérification du système. Ce qui veut dire qu'on ne s'intéresse pas à comment un composant est réalisé à l'intérieur s'il sort les données correctes au bon moment sur ses ports. A cause de ce raisonnement, l'utilisation d'une description d'un composant matériel avec quatre millions et plus de portes logiques pour la validation d'un système, semble de ne pas être très efficace. Il faut

ajouter aussi, qu'on n'a pas toujours la possibilité d'avoir une description complète du matériel dû au coût, disponibilité, ou soucis de propriété intellectuelle.

L'alternative, au lieu d'utiliser la description complète du processeur ou d'un autre module, est de modéliser seulement la synchronisation et le protocole de bus. L'utilisation d'une description au niveau de bus fonctionnel, qui modélise seulement la synchronisation et le protocole de bus, permet aux ingénieurs de générer des vecteurs de test qui affectent directement le circuit sous vérification sans le surcoût des états internes d'une description complète du matériel. En effet, elle peut facilement modéliser des cycles externes de bus sans la nécessité d'exécuter des états internes, tels que les étapes requises par exemple pour le remplissage ou l'écriture d'une ligne de cache dans le cas d'un processeur.

Le comportement d'un composant au niveau de bus fonctionnel consiste alors en une description algorithmique. La différence par rapport à une description purement fonctionnelle, est cependant, que le comportement des entrées et des sorties du bloc est précis au niveau du cycle. Par conséquent, des cycles d'attentes sont insérés dans la description algorithmique pour tracer clairement des frontières du cycle d'horloge et quand les opérations d'entrées et de sorties se produisent. Les différences principales par rapport à des descriptions au niveau RTL sont dans le comportement du composant, décrit encore algorithmiquement et dans le choix de la liste des primitives utilisées pour décrire le comportement et l'enchaînement des actions dans le temps. Par conséquent, les descriptions comportementales sont plus compactes et plus faciles à comprendre, en raison du niveau d'abstraction plus élevé. L'avantage principal des composants comportementaux décrit au niveau de bus fonctionnel est qu'ils se simulent plus rapidement que au niveau RTL.

Une des caractéristiques des composants à ce niveau est de respecter l'ordre précis des événements aux entrées et aux sorties. Ils contiennent une communication avec l'extérieur précise au niveau cycle, tandis que leur comportement reste fonctionnel et approximatif par rapport au temps d'exécution. Comme décrit précédemment, les interfaces des composants matériels sont modélisées précisément au niveau cycle avec des signaux proches du niveau RTL. Ce qui veut dire qu'à l'intérieur de l'interface d'un composant matériel au niveau de bus fonctionnel, les signaux du protocole de bus sont représentés par instanciation d'une variable correspondante à un signal ou à une groupe des signaux. Un exemple du comportement de l'interface d'un composant maître peut ressembler au code source d'exemple suivant [ARM04] :

```
void ahb0master
{
...
switch (state) // machine d'état fini d'un composant maître AHB
{
 case IDLE: // phase d'attente
```

```
      if (valid)
      {
            state = ARB;
      }
      else
      {
            state = IDLE;
            goto EXIT;
      }

case ARB: // phase d'arbitrage
      // vérification d'accès au bus
      grant = ahb0master->checkGrant(1);

      if (grant != OK) // requête d'accès au bus
      {
            ahb0master->requestAccess(1);
            goto EXIT;
      }
      if (!READY) // vérification des cycles d'attente
      {
            goto EXIT;
      }
      // alors continue
      state = ADDR;

case ADDR: // phase d'adresse

      if (ARBITRATE || !READY)
      {
      // soit on a accordé le bus ou soit nous sommes encore dans
      // la phase d'adresse parce qu'il y a eu un cycle d'attente
      // dans l'étape de données

            if (!READ) // transfert d'écriture
            {
                  ahb0master->write(addr,noData,ctrl);
            }
            else // transfert de lecture
            {
                  ahb0master->read(addr,noData,ctrl);
            }
            goto EXIT;
      }
      // alors continue
      state = DATA;

case DATA: // phase de donnée

      if (!READ) // transfert d'écriture
      {
            ahb0master->write(addr, &data, ctrl);
      }
      else // transfert de lecture
      {
            ahb0master->read(addr, &data, ctrl);
      }
break;

default :
break;
```

```
}

EXIT : // sortir
     ;
...
}
```

Ce qui décrit le vrai circuit est l'ajout de cette interface pour permettre une simulation plus précise et plus proche du comportement réel d'un circuit en gardant la rapidité d'une description algorithmique. L'exemple montre une partie de l'automate de l'interface AHB d'un contrôleur DMA décrit au niveau de bus fonctionnel. Cet automate emploie des fonctions prédéterminées comme checkGrant, requestAccess et d'autres, qui seront plus tard réalisées en matériel car ils font partie du protocole du bus AMBA. Sa machine d'état fini et ces fonctions prédéterminées, sont tous décrites en SystemC. Ce qui est l'atout du langage SystemC qui permet de décrire en un même langage le comportement du circuit et son interface fonctionnelle.

Plus en détails, la fonction checkGrant() est appelée par un maître de bus pour vérifier s'il possède l'accès sur le bus. La valeur de retour indique si le maître a obtenu le droit d'accès sur le bus, « granted ou not-granted » pour le cycle courant. Si le bus n'est pas accordé la fonction requestAccess() est appelée pour réessayer d'obtenir accès sur le bus dans le cycle prochain. Puisque les descriptions sont précises au niveau cycle d'horloge, la variable "grant" retournée correspond toujours à une requête faite dans le cycle précédent.

Les fonctions « read » et « write » sont utilisées par le composant pour effectuer des accès d'écriture et de lecture une fois qu'on lui a accordé le bus. Ces fonctions possèdent trois paramètres : ADDR, DATA et CTRL, indiquant l'adresse, les données et l'information de commande lié au transfert. Le paramètre de commande est utilisé pour passer des informations d'accès pour le transfert fait pendant le cycle d'adresse ou de données. L'information d'accès concernant tous les paramètres des signaux AHB, comme : HLOCK, HBURST, HTRANS, HPROT et HSIZE, sont codées dans ce mot de commande.

2.4. Les composants logiciels au niveau de bus fonctionnel

Le composant logiciel au niveau de bus fonctionnel inclut le programme d'application et le système d'exploitation habituellement représenté au niveau jeu d'instructions ISA (Instruction Set Architecture). A ce niveau le logiciel ne prend pas en compte le code nécessaire pour l'initialisation du processeur et la configuration de certains des périphériques car ce code est émulé par le débuggeur du simulateur de processeur.

En bref, le logiciel est représenté comme un programme ISA compilé et assemblé par rapport à certains des paramètres du matériel qui étaient fixés, mais sans tenir en compte des détails internes du processeur.

Un exemple d'un tel programme assembleur est le code suivant qui est un extrait d'un programme de test des périphériques pour le processeur ARM946E-S.

```
main
 STMFD    r13!,{r4-r6,r14}
 LDR      r4,0x30004fe8
 MOV      r0,#1
 STR      r0,[r4,#0x38]
 BL       initialise  ; 0x30005430
 LDR      r0,0x30005634
 LDR      r0,[r0,#0x10]
 TST      r0,#1
 LDR      r5,0x3000500c
 BEQ      0x300055b4
 LDR      r0,[r4,#0x38]
 CMP      r0,#9
 BGE      0x300054c0
 MOV      r1,r0,LSL #1
 SUB      r0,pc,#0x98 ; #0x3000541c
 SUB      r2,pc,#0xa0 ; #0x30005418
 BLX      _printf  ; 0x30005c9c
 SUB      r0,pc,#0x100 ; #0x300053c0
 BLX      _printf  ; 0x30005c9c
 LDR      r0,[r4,#0x38]
 ADD      r0,r0,#1
 STR      r0,[r4,#0x38]
 BL       testContinueWatchdog  ; 0x30005b58
 MOV      r6,#0
 CMP      r0,#0
 BEQ      0x3000554c
 CMP      r0,#1
 BEQ      0x3000551c
 CMP      r0,#2
 BNE      0x30005580
 LDR      r0,[r4,#0x38]
 SUB      r0,r0,#1
 STR      r0,[r4,#0x38]
 CMP      r0,#9
 BGE      0x300055c4
 MOV      r1,r0,LSL #1
```

L'exemple représente, l'initialisation des variables globales, sauf celles qui maintiennent leurs valeurs après le reset, un test pour vérifier si le chien de garde est en marche et si la réinitialisation est provoqué par lui, si non le programme va continuer avec les tests d'intégration principaux pour vérifiés le reste des périphériques connectés vers le bus du processeur et il va afficher les résultats.

2.5. La description de sortie au niveau RTL

L'étape d'intégration de matériel et de logiciel à pour but de générer une description au niveau RTL. Le programme logiciel est implémenté dans des mémoires sous la forme d'un code hexadécimal qui sera simulé en utilisant une description du processeur ciblé. La couche d'abstraction de matériel HAL utilisé dans le prototype virtuel est remplacée par une couche de logiciel détaillé contenant tout le code de bas niveau pour l'initialisation et configuration du processeur et des périphériques. La description en RTL contient aussi la réalisation finale du réseau de communication et raffinée aux fils physiques avec tous les composants nécessaires pour établir la liaison entre les modules du système.

3. L'analyse détaillée de la phase de validation par cosimulation des systèmes monopuce au niveau du bus fonctionnel

Cette section présentera les méthodes et les techniques de validation par cosimulation des systèmes monopuce au niveau fonctionnel du bus. Le premier paragraphe commencera par l'une des techniques de cosimulation, ainsi que les modèles de simulation utilisées actuellement dans l'étape de validation des systèmes monopuce, au niveau fonctionnel du bus.

Il existe deux approches principales pour la cosimulation de systèmes hétérogènes : l'approche utilisant un simulateur unifié et l'approche utilisant plusieurs simulateurs différents.

3.1. La cosimulation utilisant un simulateur unifié.

L'approche utilisant un simulateur unifié correspond à la modélisation compositionnelle. Elle consiste à intégrer les modules à simuler en une représentation globale du système. Cette méthode permet d'avoir une complète cohérence et une vérification parfaite de la définition des interconnexions. Néanmoins, cette méthode nécessite une visualisation complète de la description afin d'identifier les moyens qui permettent d'ajouter des liens pour tracer les signaux et visualiser les variables nécessaires au débogage éventuel. Elle nécessite aussi un langage de description qui peut décrire du matériel et du logiciel en même temps.

Exemple : Simulation à l'aide d'un simulateur unifié MaxSim

MaxSim [LEN05] est un simulateur unifié basée sur le langage SystemC pour la conception des systèmes monopuce. L'environnement MaxSim donne la possibilité aux concepteurs de construire un prototype virtuel en assemblant un système à partir d'une bibliothèque des composants réutilisables existants. Cette bibliothèque contient des processeurs, des composants du

bus, des mémoires, des contrôleurs d'interruption et d'autres, qui peut être enrichie avec d'autres composants faits sur demande décrits par le concepteur.

Composant : Les composants en SystemC dans MaxSim sont modélisés au cycle près. Les composants représentent des descriptions de matériel encapsulés dans une classe de C++. Ils décrivent le comportement des éléments de matériel et fournissent les ports, qui peuvent être employés pour l'interconnexion avec d'autres composants.

Les composants possèdent des paramètres, qui peuvent être configurés pendant le temps d'instanciation. Ceci permet d'implémenter des composants qui sont configurables, à partir des mémoires et des périphériques génériques. Il est important de noter que les paramètres peuvent être configurés avant que les composants soient interconnectés, ce qui permet de réaliser des composants avec un nombre de ports configurables.

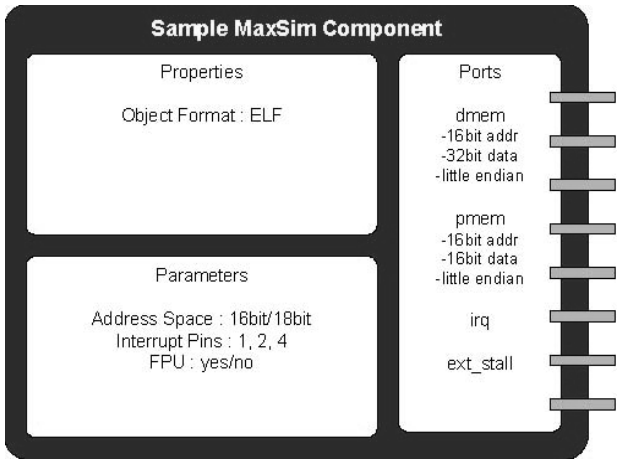


Figure 3.2 : Représentation graphique d'un composant MaxSim avec ses propriétés.

Communication : Les composants de MaxSim utilisent une communication directe, c'est-à-dire que chaque composant peut accéder aux ressources matérielles partagées d'un autre composant, en appelant directement une méthode fournie par ce composant.

MaxSim fournit deux niveaux des ports différents : signal et transaction. Les ports « signal » sont définis très près des simulateurs de matériel, simulant chaque signal indépendamment, tandis que le port « transaction » renferme tous les signaux dans un seul canal de lecture ou d'écriture. C'est un canal un peu plus abstrait qui ne peut pas être couplé directement avec une description plus fine d'un composant MaxSim.

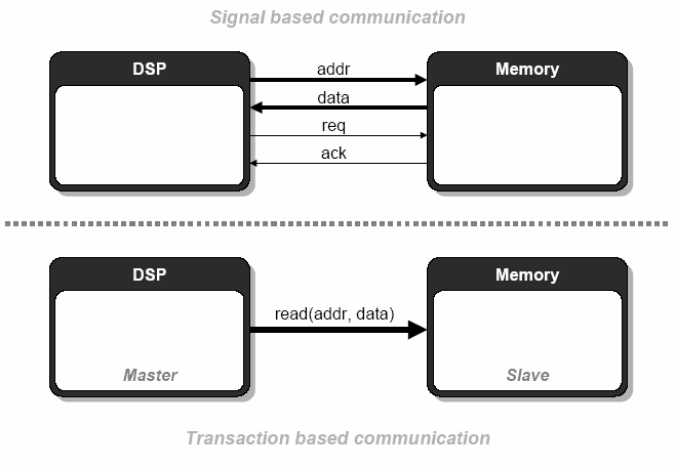


Figure 3.3 : La communication « signal » et la communication basée « transaction ».

La figure 3.3 sert à démontrer que au niveau de bus fonctionnel, l'interface de communication d'un composant peut être de type « signal » ou de type « transaction ». La différente granularité des interfaces permet de donner plus d'information sur les échanges des données entre les composants. Elle donne la possibilité de réaliser séparément, sans l'utilisation de la notion d'un canal de transaction, certains des signaux de commande comme par exemple les lignes d'interruptions. Quand même, le fonctionnement du composant reste algorithmique et l'implémentation d'une interface avec des ports « signal » nécessite d'ajouter de plus de code et nécessite respectivement plus d'efforts.

Une autre particularité pour l'outil MaxSim consiste à distinguer les ports des interfaces en ports maîtres et ports esclaves, plutôt qu'aux ports d'entrée et de sortie. Un port maître génère des transactions, tandis qu'un port esclave répond aux transactions. Dans l'exemple ci-dessus, le composant « DSP » est le maître et le composant « Mémoire » est l'esclave.

Un composant, qui a un ou plusieurs ports esclave, permet aux autres composants d'accéder sur les ressources matérielles partagées. Ceci par exemple est utilisé dans les mémoires, qui fournissent des interfaces esclaves de transaction pour des méthodes d'accès en lecture et en écriture.

Les composants, qui ont les ports maîtres peuvent accéder aux composants esclaves reliés par l'intermédiaire des méthodes d'accès bien définies. L'interface de maître de transaction fournit des méthodes de lecture et d'écriture tandis que l'interface de maître de signal fournit des méthodes pour générer et lire des signaux.

Simulation : L'environnement MaxSim supporte les deux types de simulation : la simulation événementielle et la simulation au cycle près. Les composants modélisés au cycle près seront exécutés seulement à chaque changement du front d'horloge, alors que les composants

événementiels seront exécutés par rapport aux leurs listes de sensibilité, et les événements effectifs dans le système.

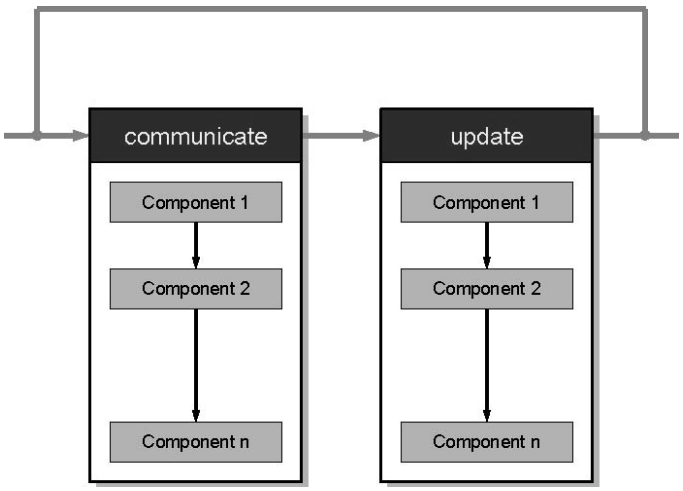


Figure 3.4 : Cycle de simulation avec les phases « communiquer » et « mise à jour ».

L'exécution de tous les composants modélisés au cycle près se produit d'une façon cyclique synchrone. Dans la plupart des cas un cycle de simulation de MaxSim sera équivalent à une horloge de matériel. Chaque cycle de simulation est divisé en deux phases : « communiquer » et « mise à jour » (voir la figure 3.4). Par conséquent, chaque composant synchronisé sera appelé deux fois par cycle.

Pendant la phase « communiquer » les composants interagissent avec les autres composants, tandis que la phase « mise à jour » est utilisée pour mettre à jour les ressources matérielles partagées et pour effectuer des écritures qui ont été demandées pendant la phase de « communiquer ».

La simulation de MaxSim peut imposer que les fonctions d'interface pour la communication s'appellent seulement pendant la phase de « communiquer ». Cependant, il ne peut pas garantir que les écritures aux ressources partagées sont reportées réellement à la phase de « mise à jour ».

3.2. La cosimulation utilisant plusieurs simulateurs différents

L'approche utilisant plusieurs simulateurs différents consiste à fournir un moteur de simulation pour chaque composant engagé dans le processus de cosimulation. La cosimulation devient un échange de données entre simulateurs dont le cheminement est tracé en accord avec les interconnexions intermodules. L'avantage d'une telle méthode permet l'utilisation d'outils existants pour réaliser la simulation et le déboguage. Néanmoins, il faut s'assurer que les simulateurs placés dans l'environnement de cosimulation permettent d'échanger leurs données de simulation avec l'extérieur via une interface de cosimulation. Cependant, cette approche implique une perte nette de performance.

Un modèle de simulation correspondant à une cosimulation utilisant plusieurs simulateurs est illustré sur la figure 3.5.

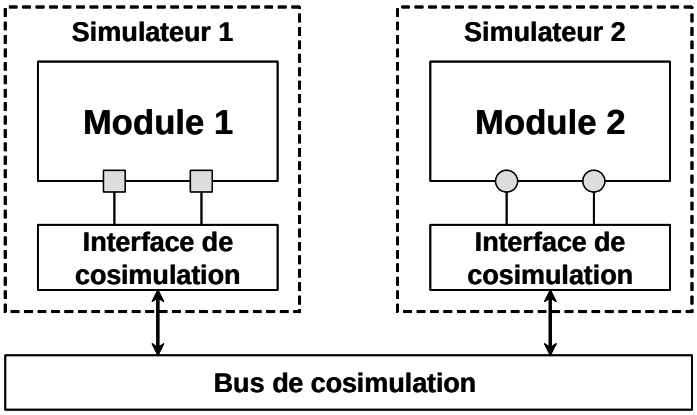


Figure 3.5 : Le modèle de cosimulation utilisant plusieurs simulateurs différents.

Les éléments de base d'un tel modèle de simulation sont :

- ✓ les différents simulateurs exécutant les composants du système à valider,
- ✓ le bus de cosimulation qui est en charge de l'interprétation des interconnexions entre les différents composants d'un système. Il est donc responsable du routage des informations entre les différents composants. Le bus de cosimulation doit rendre transparente l'existence d'un simulateur étranger dans l'instance de cosimulation.
- ✓ les interfaces de cosimulation qui facilitent la communication des différents composants du système via le bus de cosimulation. Elles sont donc en charge d'adapter chaque composant au bus de cosimulation. Le rôle de ces interfaces donc a pour but :
  - l'adaptation des différents simulateurs au bus de cosimulation – pour garantir la transmission des informations entre les simulateurs sur lesquels les composants s'exécutent et le bus de cosimulation,
  - l'adaptation des différents protocoles de communication – pour permettre l'exécution des systèmes englobant des modules caractérisés par des protocoles de communications différentes,
  - l'adaptation des niveaux d'abstraction – pour permettre l'exécution des systèmes englobant des modules situés à des niveaux d'abstraction différents.

Exemple : Simulation à l'aide d'un simulateur de processeur (ISS)

Un ISS [ARM01] est un outil qui s'exécute sur une machine hôte et qui émule la fonctionnalité d'un processeur. Il permet la simulation précise d'un logiciel qui est sensée s'exécuter sur un processeur. Un ISS modélise le processeur au niveau de son jeu d'instructions. Chaque instruction définit une relation entre les éléments internes (par exemples les registres, la mémoire interne etc.) ou externes (par exemple la mémoire externe) du processeur.

Actuellement les ISS sont utilisés beaucoup dans le processus de conception des processeurs et du logiciel. Ils font partie souvent d'un modèle de cosimulation utilisant plusieurs simulateurs, car leur rôle dans l'exploration de l'architecture, la validation et le développement du logiciel est indiscutable.

La plupart des ISS commerciaux disponibles sur le marché sont des ISS interprétatifs, qui utilisent la technique de simulation par interprétation. Un ISS interprétatif construit en mémoire une structure de données qui représente l'état du processeur cible et de la mémoire. Il entre ensuite dans une boucle d'exécution dont le corps est constitué des étapes suivantes :

- ✓ chargement : lit une instruction à partir d'un fichier contenant le programme d'application,
- ✓ décodage : analyse l'instruction et extrait le code opération (opcode) et les opérandes,
- ✓ aiguillage : utilise une instruction « switch » pour se brancher, selon le cas sur le code adéquat,
- ✓ exécution : met à jour la structure des données qui représente l'état du processeur et de la mémoire suivant la sémantique de l'instruction à exécuter.

Cette modélisation d'un simulateur logiciel à l'aide des concepts spécifiques au matériel offre une bonne précision et portabilité, mais malheureusement l'interprétation de chaque instruction limite les performances en vitesse de ce type de simulateur. Actuellement, la plupart des ISS interprétatifs exécute entre 10K et 100K instructions par seconde [ROW94].

3.3. La génération du modèle de simulation pour le prototype virtuel décrit au niveau de bus fonctionnel.

L'architecture d'un système monopuce au niveau de bus fonctionnel a deux composants principaux : (1) le logiciel d'application; (2) l'organisation de l'architecture matérielle. Elle utilise trois bibliothèques : une bibliothèque générique des fonctions logicielles (C/C++), une bibliothèque avec des composants de matériel réutilisables, contenant un ensemble de processeurs, de composants de bus et de mémoires et une bibliothèque des interfaces fonctionnelles pour la cosimulation. Ces composants sont utilisés pour la génération d'un modèle de simulation au niveau de bus fonctionnel, qui tient compte du logiciel d'application et de l'organisation de l'architecture.

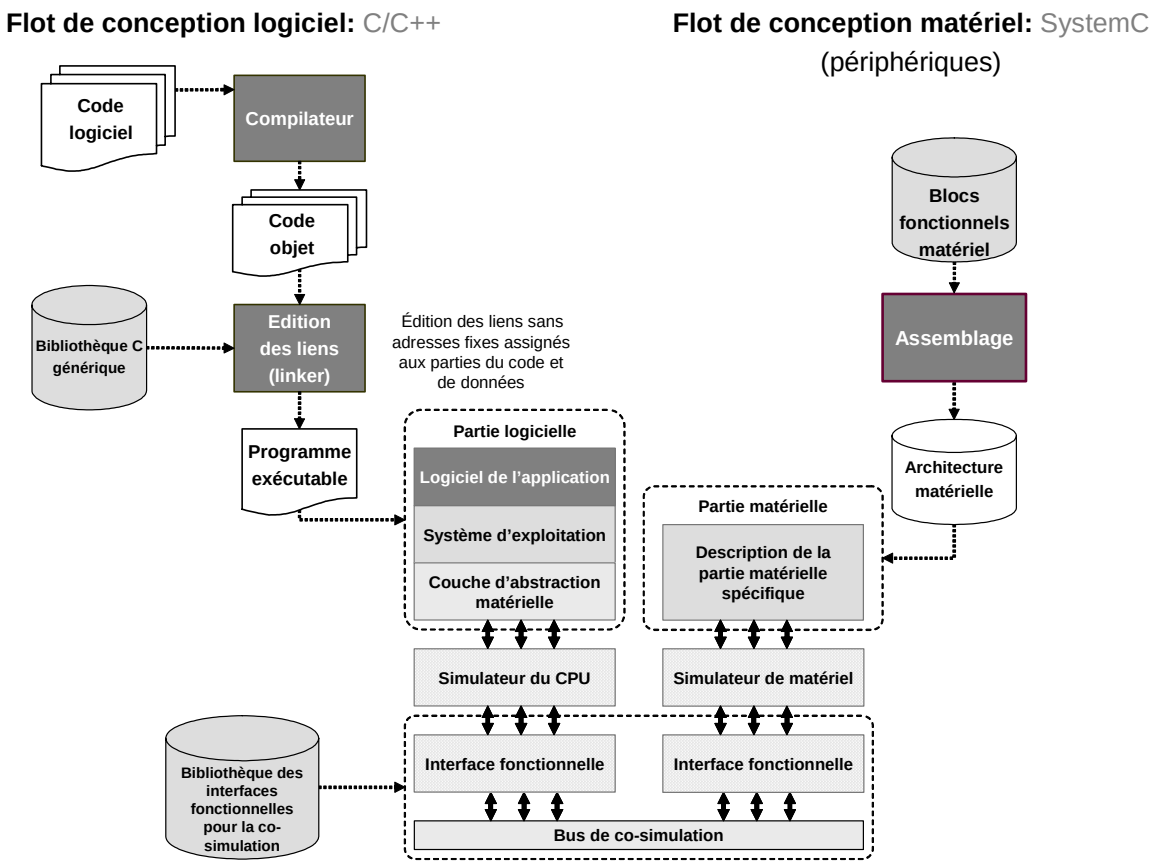


Figure 3.6 : Niveau de bus fonctionnel : La génération du modèle de simulation

La figure 3.6 montre le flot de conception pour la partie de génération de logiciel et la partie de génération de matériel de ce modèle de simulation.

Le flot de génération de logiciel est un flot classique de conception pour le logiciel embarqué. Il commence par la compilation des fichiers de code source et l'édition des liens du code objet obtenu en utilisant une bibliothèque générique des fonctions logicielles (C/C++). Ce flot est employé pour obtenir un programme assembleur exécutable au niveau ISA. Une particularité importante est qu'à ce niveau, il n'est pas pertinent d'effectuer une édition des liens avec des adresses fixes assignées au code d'instructions et au code de données pour le programme de logiciel. Nous pouvons utiliser une carte mémoire abstraite pour les adresses du logiciel du système monopuce car le simulateur de processeur n'a pas besoin de la carte mémoire effective. Le flot de conception de matériel consiste juste à assembler et vérifier des blocs fonctionnels de matériel dans un sous-système pour l'application.

Ce modèle de simulation est utilisé pour la validation et pour le déboguage de l'application. Il sert pour examiner l'exécution du logiciel d'application avec l'architecture de matériel ciblé. Ceci peut être utilisé dans une première phase de validation pour détecter des problèmes liés avec l'application telle que des tailles de données, le débordement des tampons et des problèmes de synchronisation. Si on précise certaines adresses, ce modèle aide aussi au développement du

logiciel dépendant du matériel (HdS) ou HAL pour accéder aux périphériques tels que des contrôleurs d'interruption, de contrôleur d'accès direct mémoire ou d'autres.

En utilisant ce modèle de simulation, les concepteurs peuvent explorer aussi des détails fins de l'architecture, par exemple, examiner des différentes tailles de caches ou analyser l'exécution de la communication. Les composants de bus à ce niveau permettent une estimation de la performance par profilage des conflits sur les opérations de lecture et d'écriture, qui aide à la détection des goulots d'étranglement. Certains choix architecturaux peuvent être remis en cause après cette étape (réalisation par logiciel ou par matériel d'une des tâches).

4. Le concept pour l'intégration de matériel et de logiciel à partir d'un modèle au niveau de bus fonctionnel vers RTL.

Cette section présente une méthodologie pour le raffinement du prototype virtuel vers le niveau RTL. Le prototype virtuel présenté dans les sections précédentes sera utilisé comme un modèle de référence pour les équipes de développement de logiciel et de matériel. Notre attention dans ce travail était focalisée sur le raffinement systématique de prototype virtuel décrit au niveau de bus fonctionnel vers le niveau RTL. L'objectif est de produire un prototype précis de matériel synthétisable au niveau RTL en utilisant un concept systématique pour la transformation.

4.1. Le flot d'intégration de SoC à partir du prototype virtuel

Après la validation de l'architecture au niveau de bus fonctionnel, il faut générer une description RTL à partir du prototype virtuel en appliquant des transformations nécessaires pour le logiciel et le matériel. Notre concept pour le flot d'intégration est basé sur deux étapes :

(1) La première étape est de réaliser une relocalisation séparée pour créer l'image exécutable de code du logiciel tenant compte des adresses finales employant une *technique de description de la mémoire* ou « scatter loading ». La technique de description de la mémoire aide à fournir des informations nécessaires sur l'exécution et l'implémentation de chaque pièce du code d'instruction et de données de logiciel de l'application dans les mémoires. Elle peut être utilisée aussi pour la réalisation de la logique de décodage des adresses.

(2) La deuxième étape consiste à utiliser une *description spécifique de l'architecture* décrivant la topologie de communication entre le(s) processeur(s), les mémoires et les composants de matériel spécifique. Cette description est un choix important fait par le concepteur de système au niveau de bus fonctionnel, basé sur les résultats obtenus pendant la simulation du système au plus haut niveau d'abstraction. Elle sert pour remplacer les descriptions des composants comportementaux avec des descriptions synthétisables au niveau RTL à partir d'une bibliothèque existante contenant

des composants de matériel comme par exemple des processeurs, des mémoires, des périphériques et des composants de base des bus. Cette étape est très dépendante des possibilités techniques disponibles en tant que des composants de matériel. La figure 3.7 présente le concept d'intégration de logiciel et de matériel pour le passage du niveau fonctionnel du bus au niveau RTL.

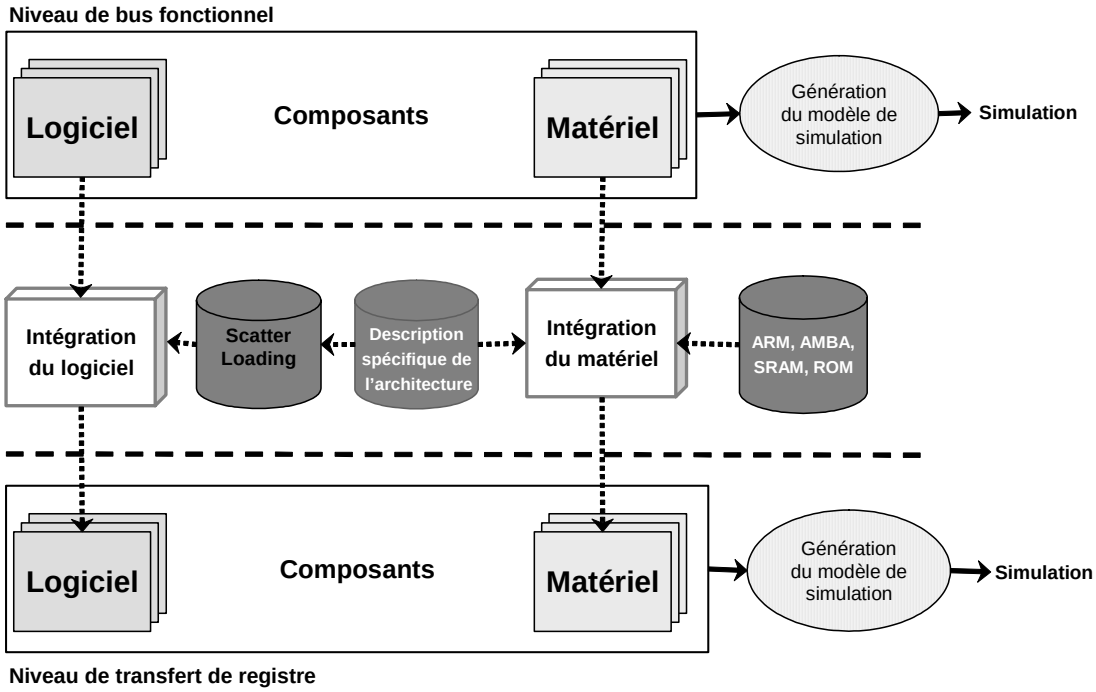


Figure 3.7 : Flot d'intégration du prototype virtuel.

La première étape dans le flot d'intégration de SoC est l'implémentation du logiciel dans les mémoires. Une relocalisation séparée doit être effectuée pour créer l'image binaire exécutable du logiciel tenant compte des adresses finales. La transformation de logiciel consiste à implémenter explicitement le logiciel embarqué dans les différents modules de mémoires et à remplacer la couche d'abstraction matérielle générique par une couche spécifique à l'architecture RTL. Pour simplifier le flot de conception, nous proposons à la phase d'intégration d'utiliser des mécanismes qui permettent d'indiquer l'organisation finale de l'architecture de la mémoire pendant le processus d'édition des liens de l'image de logiciel. Ainsi nous serons capables de décrire chaque région dans le code d'image qui a des adresses différentes dans le système de mémoire pendant le moment d'initialisation et d'exécution. Un de ces mécanismes permettant ceci est le fichier de description de la mémoire ou « scatter loading file ».

La deuxième étape représente le passage ou l'intégration de chaque sous-système du circuit au niveau de bus fonctionnel avec son homologue correspondant au niveau RTL. Cette étape est nécessaire car cette dernière représentation au niveau RTL sera utilisée pour le prototypage physique du système. Cette étape de transformation peut être réalisée en employant deux techniques différentes déjà étudiées dans des travaux de thèse de Wander Cesario [CES99],

Damien Lyonnard [LYO03] et Arnaud Grasset [GRA04]. La première technique consiste en un raffinement direct des descriptions décrites à haut niveau des composants comportementaux constituant le système. L'inconvénient de cette technique est la perte de performance et l'insertion de code non synthétisable causé par à l'imperfection des outils de conception existants. La deuxième technique repose sur l'assemblage automatique en réutilisant des composants IP. Cette technique nécessite que tous les composants matériels soient disponibles au moment de l'assemblage dans une bibliothèque, ce qui limite la réalisation d'une implémentation optimale.

Une particularité dans cette deuxième étape dans le flot d'intégration concerne le raffinement de la communication jusqu'aux signaux physiques. Car il faut tenir en compte la logique supplémentaire nécessaire pour réaliser le décodage des adresses et l'arbitrage.

4.2. L'étape d'intégration du logiciel dans une mémoire

L'intégration de logiciel consiste en la génération du fichier de description et chargement de la mémoire et l'utilisation d'une couche HAL spécifique.

4.2.1. La technique de description et chargement de la mémoire ou « scatter loading »

Dans un système embarqué simple, la carte mémoire est divisée en mémoire ROM et RAM. L'image du logiciel produite par l'éditeur de liens est divisée en deux parties. Une partie pour lecture seulement ou « read-only » RO, qui contient le code et les données pour lecture seulement, et une partie lecture et écriture ou « read-write » RW, qui contient les données initialisées et non initialisées ou initialisées à zéro ZI. Habituellement, la partie « read-only » (voir figure 3.8) est placée dans une mémoire ROM et la partie « read-write » est initialisée de la mémoire ROM à la mémoire RAM avant que l'exécution commence. Mais les systèmes monopuce utilisent souvent une carte mémoire plus complexe, qui peut être composée de plusieurs mémoires différentes comme ROM, SRAM, DRAM, FLASH et ainsi de suite.

Une image de programme se compose des régions qui peuvent occuper différents endroits au moment de chargement et au temps d'exécution. Ceci signifie que juste avant qu'une image soit exécutée, il y a quelques régions qui doivent être déplacées des endroits auxquels elles ont été au commencement chargées dans la mémoire. Par exemple, les données de lecture et écriture initialisées peuvent résider dans la mémoire ROM, mais elles doivent être copiées dans une mémoire RAM quand le programme commence à s'exécuter.

Il existe par définition deux types de régions : région de chargement et région d'exécution.

Région de chargement est la mémoire qu'un programme occupe avant qu'il commence à s'exécuter. Dans un système sur puce ce programme est déjà en mémoire au contraire des

ordinateurs classique où il serait en mémoire secondaire comme par exemple un disque dur. *Région d'exécution* est la mémoire utilisée par un programme tandis qu'il s'exécute. Une région de chargement peut contenir une ou plusieurs régions d'exécution, tandis qu'une région d'exécution appartient à seulement une région de chargement.

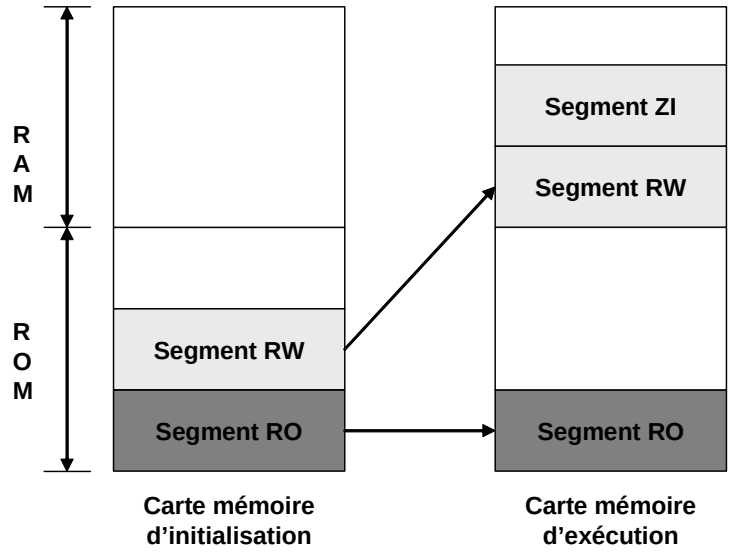


Figure 3.8 : Une carte mémoire simple montrant la place des segments en mémoire à l'initialisation (load view) ou pendant l'exécution (execute view).

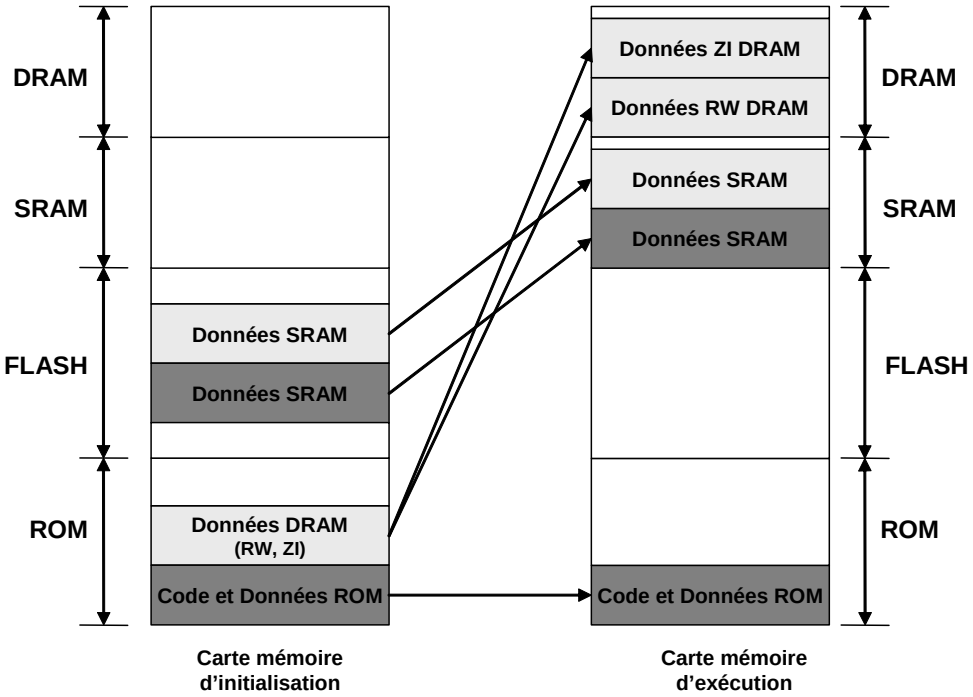


Figure 3.9 : Une carte mémoire plus complexe montrant aussi la place des segments en mémoire à l'initialisation (load view) ou pendant l'exécution (execute view).

La technique de la description de mémoire est un mécanisme fourni par l'éditeur de liens [ARM98], qui permet de diviser une image exécutable dans des régions qui peuvent être placées indépendamment dans la mémoire.

Il y a deux mécanismes disponibles pour décrire où les régions d'image devraient être placées dans la mémoire au temps d'exécution :

- ✓ En utilisant la ligne de commande avec des options -RO et -RW, pour indiquer les adresses d'exécution des régions de lecture seulement et de lecture et d'écriture. C'est la méthode la plus simple, utilisée dans les systèmes qui ont une carte mémoire simple.
- ✓ En utilisant un fichier de description et de chargement de la mémoire, qui est la méthode préférée pour des cartes mémoire plus complexes et pour les images qui ont plus de deux régions d'exécution. Cette technique permet de diviser l'image de programme dans plusieurs régions de code et de données qui peuvent être placées séparément dans la carte mémoire. L'endroit de chaque région peut différer entre le moment de chargement et le moment d'exécution, en copiant le code et les données de l'application de son adresse de chargement à son adresse d'exécution.

L'information de placement est contenue dans un fichier de description. Le fichier de description de la mémoire est un fichier texte décrivant comment les différentes parties dans une image sont assignées aux régions séparées de la mémoire. Il est utilisé pour commander le placement du code de l'application dans l'espace d'adressage de la mémoire. Ce fichier peut être utilisé aussi pour définir des adresses des composants périphériques liés à la mémoire.

Dans notre flot de conception ROSES, nous avons voulu implémenter cette technique et pouvoir générer automatiquement à partir d'un modèle « metadata » appelée COLIF [CES02] qui décrit l'architecture virtuelle de l'architecture du système sous développement. Cette automatisation fait partie des travaux de thèse de Marcio Oyamada. Elle consiste à extraire les adresses des régions à partir de deux paramètres disponibles dans les composants de mémoire correspondant à l'adresse de début et à l'adresse de fin de la région de mémoire.

4.2.2. L'utilisation d'une couche HAL spécifique

En outre, dans le flot de conception d'intégration de logiciel, la couche d'abstraction matérielle générique est remplacée par une couche spécifique à l'architecture. C'est nécessaire parce que, le code d'initialisation est spécifique pour l'application et le processeur ciblé. Dans la plupart des systèmes embarqués, une séquence d'initialisation s'exécute pour configurer le système avant d'exécuter la tâche principale. Au niveau RTL, il faut le préciser car au niveau de bus fonctionnel c'était caché par le simulateur de processeur. La séquence d'initialisation par défaut pour un processeur ARM est montrée sur la figure 3.10 [ARM02a].

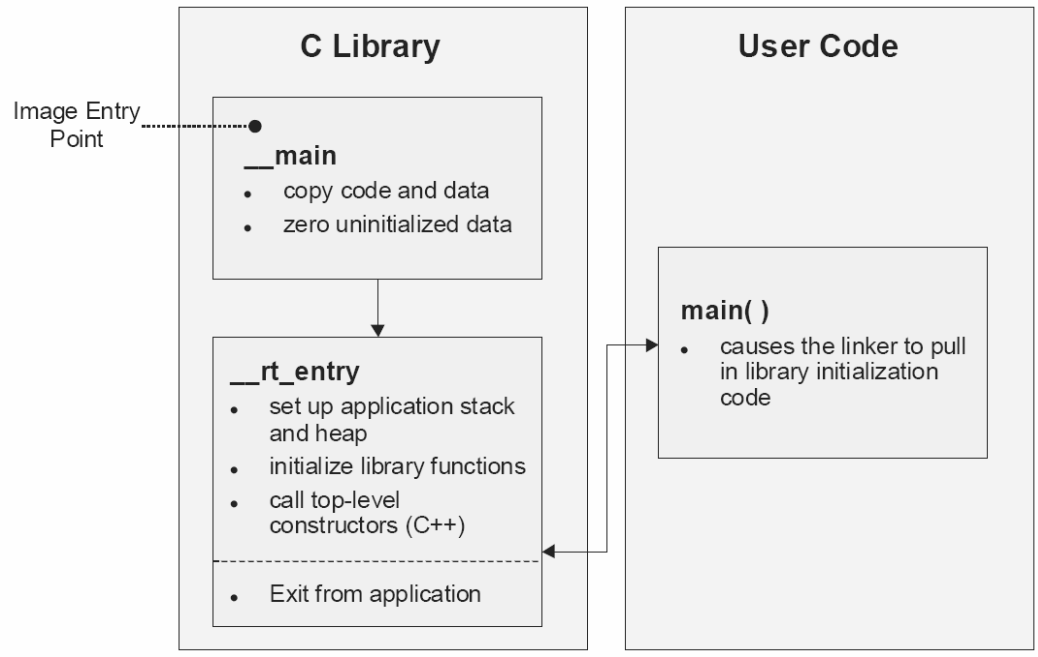


Figure 3.10 : Le programme de l'amorce d'un processeur ARM utilisant une couche HAL générique.

Nous allons détailler cette séquence d'initialisation car la plus grande partie des travaux expérimentaux de recherche de cette thèse sont réalisées autour d'un processeur ARM946E-S de la société ARM. Ce processeur, ainsi que son application seront présentés dans l'un des chapitres suivants.

Au **niveau de bus fonctionnel**, la séquence d'initialisation peut être divisé en deux blocs fonctionnels : un bloc fonctionnel nommé __main qui est responsable de l'initialisation de l'image du logiciel dans la mémoire pendant l'exécution, et un bloc fonctionnel __rt_entry responsable de l'initialisation de la bibliothèque C.

Le bloc fonctionnel __main s'implique dans l'initialisation du code dans la mémoire et dans le déplacement de la partie code et les parties données en les copiant de la région de chargement à la région d'exécution. Cette étape est significative quand l'endroit d'exécution du code et des données diffère par rapport au moment de chargement

Apres cette initialisation le __main se branche alors au bloc fonctionnel __rt_entry (entrée d'exécution). Il est utilisé pour placer les piles d'application, initialiser des fonctions de bibliothèque C et des données statiques, et appeler tous les constructeurs des objets globalement déclarés (seulement pour C++). Le bloc fonctionnel __rt_entry se branche à son tour à la fonction main(), qui est l'entrée pour l'application de l'utilisateur. Quand l'application principale main() a fini l'exécution, la fonction __rt_entry rend le contrôle de nouveau au débuggeur du simulateur de processeur.

Les piles d'application sont placées pendant l'initialisation de la bibliothèque C avec la routine __rt_entry. Nous avons la possibilité de reconfigurer le placement et la taille de la pile en ajoutant et modifiant une routine responsable de l'initialisation qui est appelé __user_initial_stackheap.

La fonction main() a un rôle significatif spécial dans la séquence d'initialisation. La présence d'une fonction main() force l'éditeur de liens à lier dans le code d'initialisation les fonctions __main et __rt_entry. Sans main(), la séquence d'initialisation n'est pas liée dedans, et en conséquence, certaines fonctionnalités standards de bibliothèque de C ne sont pas soutenues.

Jusqu'ici, nous avons supposé que l'exécution commence avec la fonction __main, et que c'est le point d'entrée de la routine d'initialisation de bibliothèque C. En fait, n'importe quelle application embarquée réelle exécute une initialisation au niveau du système au démarrage. Habituellement il effectue l'initialisation et la configuration de processeur, l'initialisation de la mémoire et le « remap », chargeant les différentes régions d'exécution de la mémoire exigée par le code C de l'application, activation des caches et des interruptions. Le paragraphe suivant discute ceci plus en détail.

Dans **le niveau RTL**, nous sommes obligé d'ajouter une routine nommée « reset_handler », qui s'exécute immédiatement au démarrage de système Nous avons également ajouté une routine de __sub_main, qui s'exécute juste avant la fonction main() de l'application principale. Le reset_handler est un programme (codé en assembleur) qui est exécuté juste après le reset du système. Comme minimum, le reset_handler initialise les pointeurs des piles pour les différents modes d'exécution de l'application. Pour des processeurs avec des sous-systèmes mémoire locaux (c'est-à-dire avec des caches et/ou des mémoires Tightly Coupled Memory) une certaine configuration doit être faite à ce stade dans le processus d'initialisation. Ce sont des routines pour la configuration des composants comme Memory Management Unit ou Memory Protection Unit et pour l'activation des caches et des mémoires TCM. Une fois que l'initialisation de toutes ces routines est effectuée, le reset_handler se branche typiquement à la routine __main pour commencer la séquence d'initialisation de bibliothèque C.

Il y a quelques parties (composants) d'initialisation de système par exemple l'autorisation des interruptions, qui sont généralement effectué après que l'initialisation de la bibliothèque C est faite. Le code de la routine __sub_main accomplit ces tâches avant que l'application principale commence à s'exécuter.

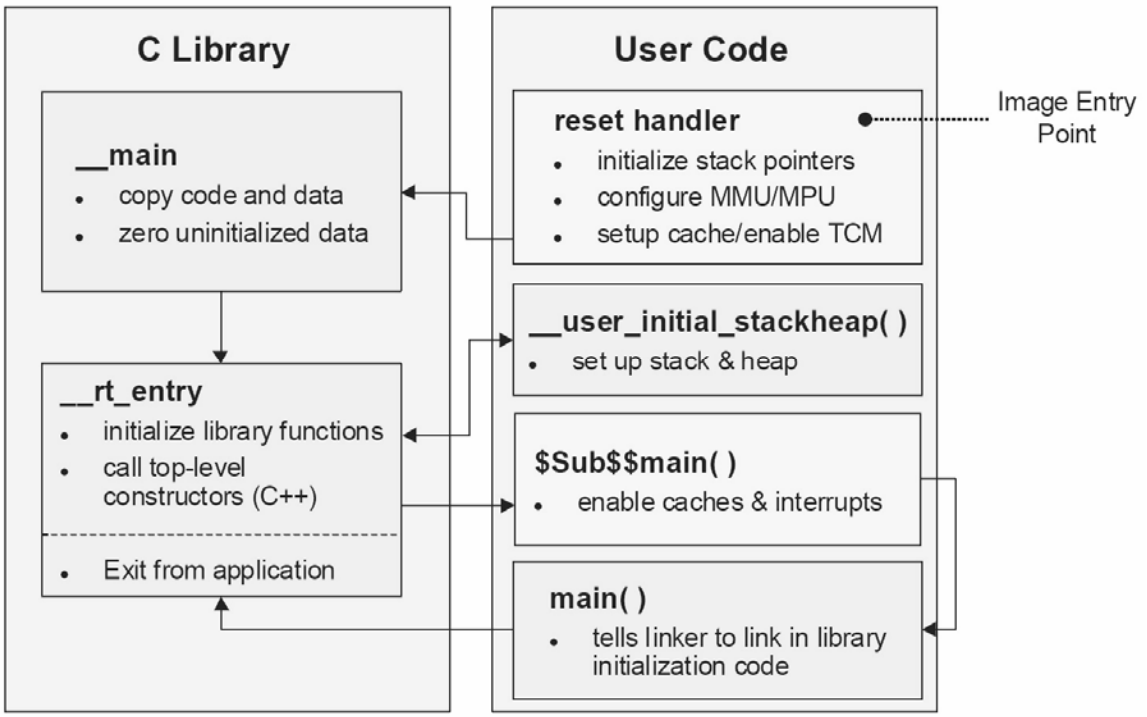


Figure 3.11 : Le programme d'amorce d'un processeur ARM utilisant une couche HAL spécifique.

La figure 3.11 montre un schéma fonctionnel du programme d'amorce d'un processeur ARM et donne une vue d'ensemble des composants dans la couche d'abstraction matériel spécifique. Il est encore développé manuellement par le concepteur de système car c'est une tâche difficile pour l'automatisation.

4.3. Etape d'intégration et de raffinement des composants de matériel

L'étape d'intégration comporte un ensemble des tâches qui sont nécessaires pour assembler des composants de matériel pour remplir les conditions et les exigences d'un système. Comme montré à la figure 3.12, l'étape d'intégration prend comme entrée une description spécifique de l'architecture matérielle et un ensemble des composants matériels qui ont été choisis pour implémenter l'architecture du système. La sortie de cette étape est une architecture, où les composants de matériel sont décrits au niveau RTL entièrement synthétisable.

Il y a trois approches principales pour l'intégration des composants matériels, comme illustré sur la figure 3.12 [WAG04]. Cette classification est basée sur une séparation entre la partie de la communication et la partie du calcul pour chaque composant.

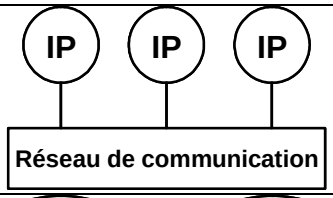
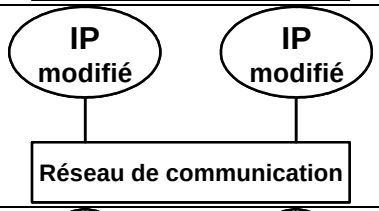
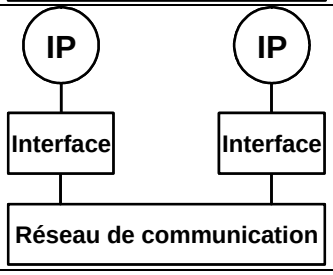
<table>
<tr><th colspan="2">Type des composants</th><th>Approche</th><th>Matériel</th></tr>
<tr><td colspan="2">Composants homogènes</td><td>Par assemblage</td><td> 
</td></tr>
<tr><td rowspan="2">Composants hétérogènes</td><td>Le code source est disponible</td><td>Par dérivation</td><td> 
</td></tr>
<tr><td>Le code source n'est pas disponible</td><td>Par synthèse</td><td> 
</td></tr>
</table>

Figure 3.12 : Approche pour l'intégration des composant matériel.

L'approche par assemblage basé autour d'un standard de communication

Dans l'approche basée autour d'un standard de communication, les interfaces des composants sont conformes à une norme donnée, de sorte qu'elles se correspondent directement entre elles ou à une structure standard de communication, et leurs fonctionnalités de calculs correspondent exactement aux fonctionnalités désirées par le système monopuce. Dans ce cas-ci, les composants peuvent être intégrés dans un système sans avoir besoin d'une adaptation fonctionnelle. Les approches par assemblage utilisent une spécification du système, à partir de laquelle on extrait un certain nombre de paramètres tels que les protocoles. Des composants de bibliothèques sont alors sélectionnés, configurés et assemblés pour générer l'architecture du système. L'efficacité de ces méthodes est limitée par leur besoin de bibliothèques de taille importante ainsi que par l'utilisation d'un seul modèle d'assemblage. L'inconvénient principal de cette approche est qu'il existe autant des normes de communication que de fabricants des composants matériels. Chaque société utilise ses propres standards, ce qui impose des obstacles devant la réutilisation des bibliothèques des composants matériels déjà développés pour une autre norme.

L'approche de dérivation des composants IP ou l'approche par synthèse :

L'approche de dérivation des composants IP de matériel peut être utilisée seulement quand le code source des composants est disponible et peut être directement modifié, de sorte qu'un nouveau composant puisse être dérivé du précédent. Dans ce cas, une modification sera nécessaire pour adapter les interfaces hétérogènes ou pour faire correspondre le comportement du composant

calcul à la fonctionnalité exigée par le système monopuce. Les approches par synthèse utilisent une description algorithmique du composant à partir de laquelle est extraite une machine d'états finis qui représente la partie contrôle du circuit. Ces méthodes offrent un haut niveau d'abstraction pour la conception des composants mais leur conception reste manuelle ou restreint les composants générés. Cela les rend mal adaptés à la conception des systèmes monopuces complexes.

Si le code source des composants n'est pas disponible pour des raisons de propriété intellectuelle, l'intégration directe des composants de calcul à la fonctionnalité du système n'est pas possible. Dans ce cas-ci, l'intégration doit être réalisé par l'implémentation des nouveaux composants dans le système. Si l'ensemble des composants à intégrer a des interfaces hétérogènes, il faut utiliser une approche basée sur la synthèse des interfaces de communication, où les adaptateurs sont générés automatiquement et insérés entre les composants.

L'approche par synthèse des composants matériel et par synthèse de la communication :

L'utilisation du concept de synthèse des composants IP de matériels et la synthèse de communication est la troisième approche différente pour résoudre le même problème de l'intégration entre les composants IP hétérogènes, qui ne suivent pas des normes standard. La synthèse comportementale des composants IP matériel est une solution habituellement basée sur le concept orienté objet, venant de la communauté du logiciel. Elle peut être appliquée seulement pour l'intégration des composants de matériels appelés « soft », un exemple d'un tel composant est le processeur Xtensa de la société Tensilica. La synthèse des composants IP matériels est une méthode essentiellement automatique guidée par le concepteur. D'autre part, la synthèse de la communication, utilise les concepts de raffinement automatique de haut niveau de la logique d'interconnexion. La synthèse de communication est un processus automatique, sans l'interposition du concepteur.

Bien que de nombreux travaux aient déjà été réalisés, la génération automatique des interfaces reste d'autant plus difficile que la communication devient de plus en plus compliquée. A notre avis, les points suivants posent problème : le manque de flexibilité des méthodes existantes, le besoin de supporter des services de communications plus complexes, et l'intégration logiciel/matériel. La communication entre le logiciel et le matériel est difficile car elle implique aussi bien les pilotes de communication que l'interface de communication, le processeur et ses mémoires.

4.3.1. Intégration des composants matériels

Pour notre flot d'intégration la technique que nous avons choisie est l'assemblage des composants matériels combiné avec une approche basée aussi par assemblage des interfaces de la

communication car nous pensons qu'une telle approche est mieux adaptée pour la réalisation des systèmes complexes. Les objectifs de ce flot d'intégration sont de :

- ✓ diminuer le temps de conception des bibliothèques
- ✓ diminuer le temps d'introduction d'un nouveau composant
- ✓ cibler les interfaces aux besoins de l'application
- ✓ supporter des services de communication
- ✓ faciliter l'intégration logiciel/matériel

Actuellement l'assemblage de matériel à partir du modèle de prototype virtuel vers le modèle RTL est faite à la main. Quelques outils comme l'outil « Platform Express »de Mentor, l'outil « coreTool » de Synopsys et l'outil « ASAG » de TIMA fournissent un chemin pour l'intégration de matériel produisant des descriptions nécessaires au niveau RTL à partir du niveau de bus fonctionnel. Dans ces outils de conception les composants sont produits par la personnalisation des composants de base déjà existant dans une bibliothèque. Les composants dérivés sont configurées pour remplir les conditions d'application et diffèrent normalement en termes de modules de mémoire et des composants périphériques rajoutés. D'autres outils fournissent des méthodes de raffinement de TLM jusqu'aux signaux physiques, basé sur des protocoles de bus tels que le standard AMBA.

La méthode d'assemblage et les outils de génération automatique des interfaces de matériel/logiciel utilisées pour le raffinement des interconnections physiques ne sont pas présentés car ils font partie des travaux de thèse de Arnaud Grasset.

5. Conclusion

Dans ce chapitre nous avons présenté notre concept pour l'intégration de la partie matérielle et la partie logicielle à partir du modèle d'un système monopuce modélisé au niveau de bus fonctionnel vers le niveau RTL. Nous avons discuté chaque étape d'intégration, ainsi que les techniques qui sont utilisées. Nous avons également présenté les descriptions d'entrée et de sortie de ce niveau, en donnant une définition sur les propriétés des descriptions qui caractérisent leurs composants.

Le point faible de cette approche est le manque de l'automatisation pour certaines étapes d'intégration, mais c'est un objectif devant nous, que nous somme en train de développer.

Cette approche semble être une manière efficace pour l'étape d'intégration des systèmes sur puce car les avantages qu'elle propose sont : une systématisation du processus de génération basée sur l'assemblage de l'architecture à partir des composants matériels préconçus et une diminution du temps pour l'intégration du logiciel.

Chapitre 4

LA METHODOLOGIE DE CONCEPTION DE NIVEAU RTL AU NIVEAU PHYSIQUE : LE PROTOTYPAGE PHYSIQUE

Dans ce chapitre, nous avons étudié et examiné les différents modèles et méthodologies concernant la conception des systèmes monopuce à partir du niveau RTL. Nous allons également étudier les particularités des composants décrits en SystemC RTL, et nous proposerons un modèle de codage pour optimiser la génération automatique de ces descriptions.

1. Introduction

Le prototypage physique représente les étapes ultimes d'une chaîne de développement d'un système monopuce. Son résultat représente l'obtention des dessins des masques servant à la fabrication du circuit intégré. La méthodologie de conception physique consiste en plusieurs étapes consécutives de conception telles que la synthèse et la vérification logique, le placement et le routage automatique, la synthèse automatique d'arbre d'horloge, la distribution et la gestion de l'alimentation, la vérification de règles de dessin, etc. Dans ce chapitre, il sera présenté seulement les étapes de la synthèse et de la vérification logique, ainsi que leurs descriptions d'entrée et de sortie. Le type de conception traité s'applique essentiellement sur la conception des circuits de type ASIC à base de cellules standard précaractérisées. En effet le processeur et les mémoires utilisées dans un système monopuce sont fournis par le constructeur.

2. La méthode utilisée pour la conception au niveau RTL

Une description est une représentation abstraite d'un circuit physique dont on ne conserve que les aspects essentiels pour une certaine utilisation. Une description sert donc à analyser le comportement du circuit, pour l'extraction et la vérification de propriétés. La même description est utilisée aussi à l'étape de la synthèse, pour la dérivation d'une description plus détaillée en fonction de contraintes de conception.

L'analyse dans l'étape de conception physique représente le processus dans lequel la description est décrite sous une forme exécutable pour un logiciel de simulation. La description est soumise à un ensemble de stimuli et le logiciel calcule la ou les réponses à ces stimuli. Un exemple est la simulation logique qui permet de calculer l'évolution temporelle de signaux dans un circuit logique en fonction de signaux appliqués aux entrées primaires du circuit.

Il existe encore un deuxième type d'analyse dans la conception physique qui est réalisé par des processus basés sur des techniques de preuve ou de vérification. Il n'est pas nécessaire d'appliquer des stimuli dans ce cas. Un exemple est le calcul des délais entrées/sorties d'un circuit logique par accumulation des délais des portes logiques et des interconnexions des chemins entre les entrées primaires et les sorties primaires du circuit.

Le processus de synthèse réalise des transformations sur une description de manière à dériver une nouvelle description plus détaillée et optimisée en fonction de contraintes imposées (p. ex. surface minimum, délais minimums). Un exemple est la dérivation d'un circuit logique capable de réaliser la ou les fonctions décrites par un algorithme tout en satisfaisant des contraintes sur les

types et les nombres d'opérateurs disponibles (p. ex. N additionneurs et M multiplieurs) et sur le temps de cycle.

2.1. La description d'entrée au niveau RTL

La partie matérielle d'une description RTL est décrite sous la forme d'une représentation synchrone du système décomposé en une partie de contrôle et une partie opérative travaillant de manière concurrente. Il est habituellement implémenté en terme de registres, circuits combinatoires, bus de bas niveau, et circuits de commande. Les composants de base sont des modules logiques complexes, tels que ALU, multiplexeur, décodeur, registre, etc. L'information est constituée par des bits et des mots et le temps est réduit à des coups d'horloge. La partie logicielle d'un système au niveau RTL est décrite comme un programme en langage machine, comme au niveau de bus fonctionnel, mais en plus le processeur est aussi décrit au niveau RTL. Le temps peut être explicite (réduit à des coups d'horloge) ou implicite (les événements sont gérés par la causalité).

Le but d'une description RTL est de développer et tester l'architecture interne et la logique de contrôle d'un circuit intégré de telle sorte que la conception satisfasse les fonctionnalités exigées et les contraintes de temps par l'application. Il est également utilisé pour spécifier la conception d'un circuit dans une formation indépendante de la technologie destinée à la synthèse automatique. A ce niveau le processus de conception est appelé conception niveau RTL ou synthèse RTL.

2.2. La description de sortie au niveau logique

Une description au niveau porte logique (gate level) décrit un composant en termes de fonctions logiques booléennes et des éléments mémoire simples tels que des bascules. La correspondance entre équations booléennes et portes logiques est immédiate. Les expressions logiques peuvent être transformées dans les formes fonctionnellement équivalentes avant leur implémentation réelle en portes logiques. Le comportement des blocs en logique booléenne implante des fonctions booléennes simples telles que : NAND, NOR, etc. A ce niveau le processus de conception est appelé conception au niveau logique.

3. L'étape de conception au niveau RTL

La figure 4.1 illustre le flot de conception basé sur VHDL. On distingue les étapes suivantes :

- ✓ Création des descriptions VHDL (édition textuelle). Il s'agit de créer des descriptions VHDL synthétisables au niveau RTL. Les fonctions complexes sont décrites de manière

comportementale. Typiquement, une partie de contrôle serait décrite comme une machine à états finis (FSM) et une partie opérative, p.ex. une unité arithmétique et logique (ALU), serait décrite par ses équations logiques ou arithmétiques.

- ✓ Simulation des descriptions avant synthèse. Il s'agit de créer un ou plusieurs séquences de test en VHDL pour valider la description RTL du circuit.
- ✓ Synthèse des descriptions. La synthèse détermine une réalisation de la description RTL à base des cellules standards (portes logiques combinatoires et bascules séquentielles). La synthèse est usuellement gouvernée par un ensemble de contraintes définies séparément (surface, délais, consommation). La synthèse se base aussi sur une bibliothèque de cellules standard spécifique à une technologie donnée (par exemple 0,12 μm CMOS) qui définit entre autres pour chaque cellule sa fonction, sa surface, ses délais internes, sa consommation et ses contraintes d'environnement. Le résultat de la synthèse est une description des instances de portes et de leurs interconnexions (netlist) qui peut être générée en VHDL et en Verilog. Le premier format sera utile pour la simulation après synthèse. Le second format sera utile pour passer à l'étape de placement et routage. Il est aussi possible de générer un fichier SDF (Standard Delay Format) contenant les délais des portes.
- ✓ Simulation après synthèse. Les vecteurs de test peuvent être réutilisés pour simuler la description VHDL au niveau porte logique avec le fichier de délais obtenu après synthèse. Les descriptions VHDL (en format VITAL) des portes sont disponibles dans une bibliothèque associée à la technologie utilisée.
- ✓ Placement et routage. L'étape de placement et routage détermine une réalisation géométrique (layout) de la description au niveau portes en fonction des paramètres de la technologie considérée et d'une bibliothèque de descriptions géométriques de cellules standard. Toutes les cellules ont la même hauteur de manière à pouvoir les aligner par rangées. Le routage des interconnexions se fait usuellement par-dessus les cellules, ce qui permet de joindre les rangées et de gagner ainsi de la place. Le résultat du placement et routage est une description géométrique au format GDS2, un format standard. Il est aussi possible de générer un fichier SDF contenant les délais des portes et des interconnexions. Si la netlist n'a pas changé durant cette étape (p.ex. à cause de l'insertion d'un arbre d'horloge), le fichier SDF peut être utilisé avec la description Verilog généré après synthèse. Sinon, il faut générer une nouvelle description Verilog à partir du résultat du placement et routage.

- ✓ Simulation après placement et routage. Il s'agit ici de simuler une description Verilog avec les informations de délais extraites du layout.
- ✓ Intégration du bloc dans le système. La description layout du bloc peut être ensuite intégrée dans le reste du système à réaliser. Un outil de placement et de routage de blocs est usuellement utilisé.

Ces étapes, avec les noms des outils que nous avons utilisés dans nos expérimentations, sont présentées sur la figure suivante.

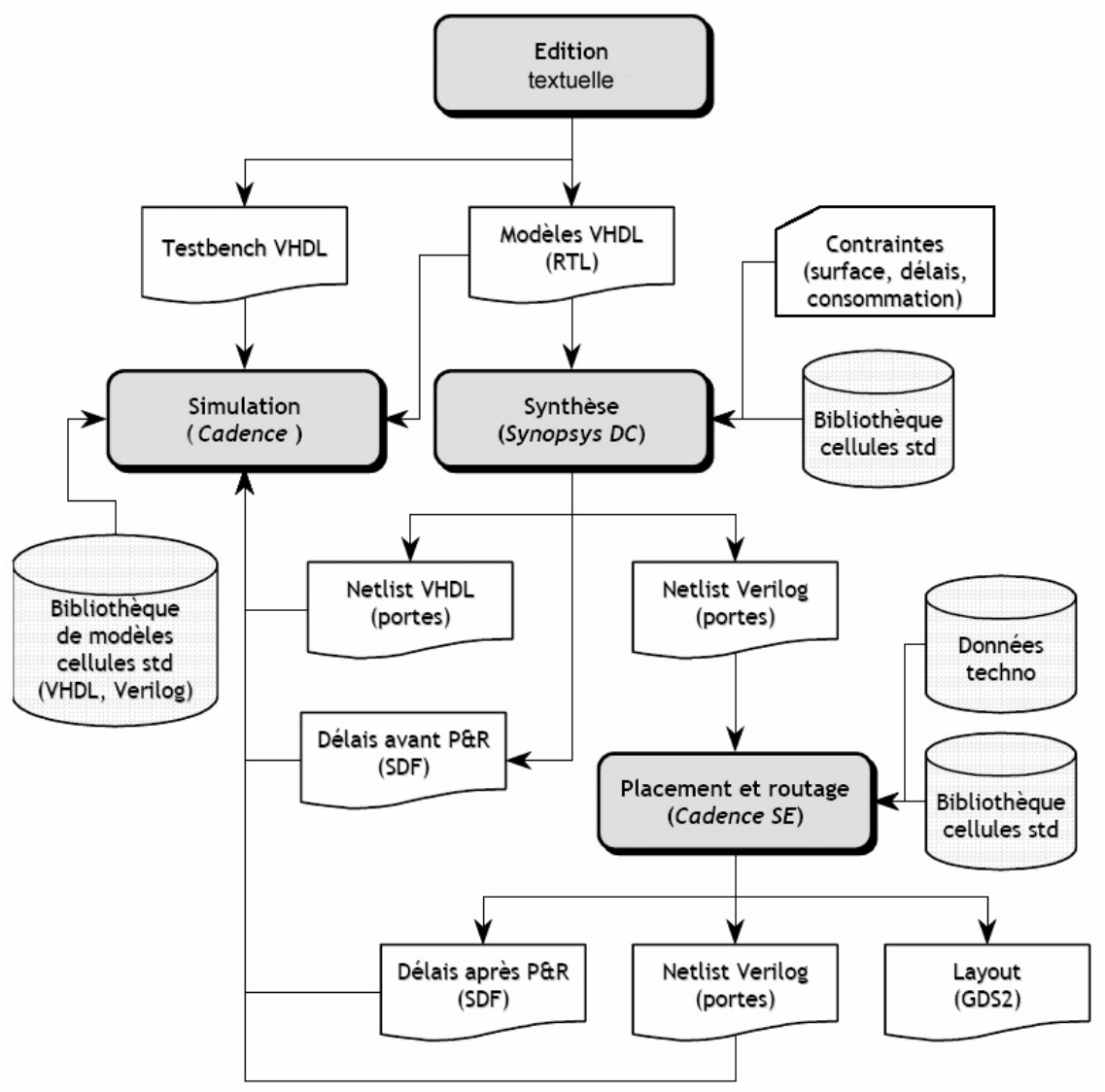


Figure 4.1 : L'exemple du flot de conception basé sur VHDL ou Verilog.

4. L'étape de la vérification par simulation.

Par leur complexité les systèmes sur puce impliquent le travail de plusieurs équipes, pour la conception et l'intégration des composants. La simulation doit donc faire face à l'hétérogénéité des

langages de codage comme VHDL, Verilog, SystemC et des composants IP, mais aussi aux différentes vues de représentation.

4.1. Les entrées pour la simulation

Les entrées pour la simulation comprennent le code de la description d'un bloc ou de l'ensemble des blocs d'un système et les stimuli sur lesquelles s'effectuera la simulation. A ce niveau de la conception, le code est spécifié en HDL (VHDL, Verilog ou SystemC RTL). Des simulateurs [CAD04] permettent aussi une spécification mixte comprenant à la fois du code VHDL et du code Verilog ou du code SystemC.

La présence des deux langages proprement dits VHDL et Verilog, permet d'optimiser la simulation en fonction du niveau d'abstraction ciblé. Par exemple au niveau porte, le standard VITAL [IEE01] doit normalement permettre une simulation efficace du VHDL. Cependant pour des conceptions de plusieurs millions de portes, l'utilisation de Verilog s'avère plus efficace. Mixer plusieurs langages d'entrée permet de profiter pour chaque description du langage le plus efficace pour la simulation. La plupart des simulateurs autorisent aussi l'utilisation des stimuli décrit en C/C++ pour accélérer la simulation. Leur utilisation peut être guidée à l'aide des scripts en TCL (shell scripting language) [TCL05].

4.2. Le modèle de simulation

Les simulateurs utilisent généralement deux types des descriptions pour le modèle de simulation. Les descriptions des composants crées au niveau RTL par l'ingénieur, c'est-à-dire que le code source du module est disponible et synthétisable et les descriptions des composants IP, qui sont fournis comme une bibliothèque de bloc souvent sous le format « SwiftModel » [SYN05]. Il est possible d'annoter les caractéristiques temporelles de ces composants à l'aide de SDF (Standard Delay Format) [SDF].

Les composants fournis sous le format « Swift », sont constitués de deux API (Application Program Intreface) : un API pour la simulation qui donne le comportement externe du bloc et un API qui sert de noyau pour les services nécessaires à la gestion des événements internes à la description. Le simulateur pour les modèles « Swift » était développé par Synopsys tandis que le noyau est proposé par Cadence. La figure 4.2 donne un exemple de la structure de ce modèle.

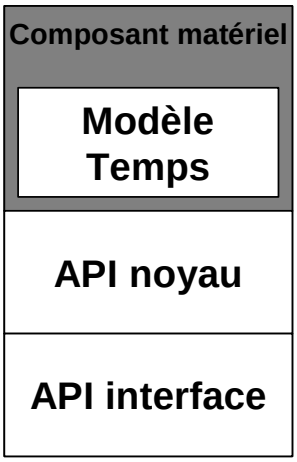


Figure 4.2 : L'exemple d'un composant utilisant le format « Swift ».

4.3. La simulation et l'accélération de la simulation

La simulation est opérée à partir du code qui est, soit interprété, soit compilé pour une plateforme de simulation ciblé. Lors de la compilation des techniques classiques d'optimisation de code sont mises en œuvre (propagation de constantes, élimination de code mort, élimination des sous expressions communes, déroulage de boucle, mise en ligne de fonctions) pour accélérer la simulation. Les algorithmes de simulation sont basés sur les cycles d'horloge ou les événements temporels. Le second cas est plus précis et permet de détecter par exemple des aléas « glitch », mais ils sont plus lents que les simulations sur les cycles. Pour la simulation utilisant des descriptions décrites en Verilog, la propagation des événements est effectuée durant la simulation. La plupart des simulateurs acceptent les compilations incrémentales, permettant ainsi d'accélérer la simulation après modification partielle du code.

Lorsqu'il s'agit de simuler un code mixte, VHDL et Verilog ou SystemC on trouvera deux techniques pour la gestion de la simulation :

- ✓ soit une technique consistant à compiler séparément chaque code (VHDL, Verilog ou SystemC) et à ajouter un noyau pour la gestion des communications entre les simulateurs,
- ✓ soit une technique effectuant une compilation native du code et générant un noyau unique pour la gestion des événements (technique INCA (Interleved Native Compiled Architecture) utilisée pas Cadence [CAD04]).

Si la simulation consiste à vérifier d'une manière exhaustive le code d'une description, il est donc nécessaire de prendre en mesure la couverture du code par les vecteurs de test. Une telle analyse est souvent intégrée à l'outil de simulation ou parfois comme un outil complémentaire. La couverture de code concerne un bloc particulier ou l'ensemble du système.

4.4. L'analyse de la simulation

L'analyse des résultats de la simulation est réalisée par trois différentes manières. La première manière c'est en comparant deux formes d'onde, la comparaison peut être continue sur une horloge. La comparaison peut aussi porter sur le contenu de registres particuliers. La deuxième manière en utilisant des points d'arrêt et de comparer de nouveau les formes d'onde structurées. La troisième manière d'analyse est portée sur les stimuli, en regardant quels sont les signaux non utilisés dans la liste de sensitivité, le code des vecteurs de test non nécessaires etc. On cherche alors à simplifier les tests opérés tout en conservant la couverture maximale du code.

5. La covérification de SystemC et de HDL

La cosimulation du code de SystemC et de HDL (Verilog ou VHDL) est extrêmement importante dans le flot de conception global, car beaucoup des systèmes sur puce sont conçus en utilisant des composants décrits en même temps en HDL et en SystemC. Alors, il est nécessaire souvent d'importer un module qui a été développé en SystemC dans un environnement HDL existant ou au contraire.

Actuellement, il n'y a aucun outil industriel pour vérifier efficacement l'équivalence entre deux descriptions d'un même composant décrit en SystemC et en HDL. Pour s'assurer que le code dérivé de HDL est exactement identique à celui codé en SystemC, la cosimulation entre les deux environnements est nécessaire.

5.1. L'environnement de cosimulation en SystemC

La plupart des outils de cosimulation entre un composant décrit en SystemC et un composant décrit en HDL génèrent un ensemble d'interfaces qui permettent à l'environnement de SystemC de communiquer avec un simulateur de HDL. Pendant la simulation, l'outil de cosimulation est responsable de :

- ✓ La synchronisation entre le simulateur de SystemC et le simulateur de HDL,
- ✓ Le transfert des données entre les deux environnements.

Il existe deux manières d'exécuter la cosimulation, le mode d'importation et le mode d'exportation.

En mode d'importation, l'outil de cosimulation génère des interfaces de communication pour importer un module HDL dans l'environnement de SystemC. Le simulateur de SystemC est le maître et le simulateur de HDL est l'esclave. L'autre mode en exportation consiste en génération

des interfaces de communication pour exporter un module de SystemC dans l'environnement de HDL. Le simulateur de HDL est le maître et le simulateur de SystemC est l'esclave dans ce cas.

5.2. La communication entre les deux environnements.

Un module simple ou un groupe de blocs peut être simulé décrit en HDL en employant un environnement de simulation HDL, tandis que le reste d'un système peut être exécuté en SystemC. Pour réaliser cette cosimulation, il faudra établir la communication entre les deux environnements. La figure 4.3 montre un exemple d'interface de cosimulation de HDL et de SystemC utilisé dans le flot de conception ROSES.

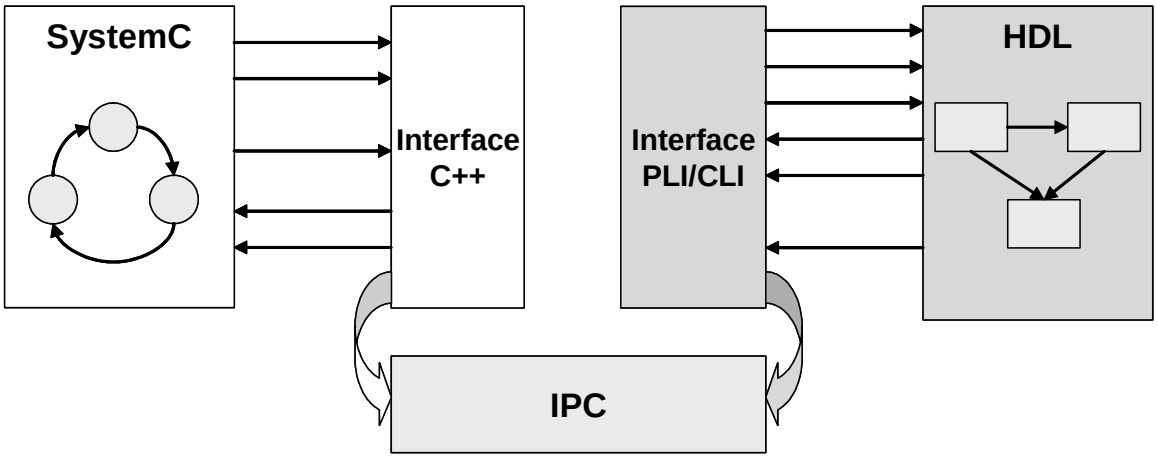


Figure 4. 3 : L'environnement de covérification de HDL et de SystemC.

Sur la figure, une partie des composants sont décrits en SystemC, en tant que des descriptions comportementales. Les signaux d'entrée/sortie de tous les blocs SystemC communiquent à travers l'environnement de simulation via une interface C++. Cette interface utilise des appels fournis par la bibliothèque IPC (Inter-Process Communication) [IPC] pour communiquer avec l'environnement de simulateur HDL.

L'environnement de simulation HDL comprend l'instanciation des modules décrits en RTL et une interface fournissant des appels spécifiques PLI (Programming Language Interface) qui utilise aussi la bibliothèque IPC pour communiquer avec l'interface C++. Les deux interfaces forment un canal de communication qui est utilisé pour transférer des données et des événements de simulation et de synchronisation entre les deux environnements. Ce canal de communication est souvent nommé bus de cosimulation.

La vitesse d'exécution de la cosimulation peut diminuer dûe au niveau d'abstraction utilisé pour la communication entre l'environnement de simulation HDL et l'environnement de SystemC basé sur la technique IPC (Inter Process Communiation).

6. L'étape de la synthèse

La synthèse logique est la transformation d'une description RTL en un réseau de portes logiques interconnectées qui réalise les fonctions souhaitées.

Le système à concevoir est décomposé en blocs d'éléments en logique combinatoire et de mémoire. Le comportement des éléments en logique combinatoire est décrit par des fonctions booléennes. Leur optimisation et le ciblage vers une structure matérielle au niveau porte sont la tâche de la synthèse au niveau logique. Le ciblage technologique consiste ensuite à transformer la liste d'interconnexion (netlist) générique obtenue après la synthèse logique en une netlist logiquement équivalente, mais ciblée sur une technologie donnée grâce à l'utilisation d'une bibliothèque spécifique. Cette bibliothèque technologique contient des logiques standard qui font par exemple la fonction y = ab + cd +fg dans le cas d'une bibliothèque visant la conception des circuits ASIC, ou des LUT et des multiplexeurs pour la bibliothèque visant le prototypage en utilisant des FPGA.

Le ciblage technologique n'est pas limité au niveau logique. Quelques systèmes réalisant le ciblage de technologie bien après des optimisations RTL (le ciblage des unités fonctionnelles en macro bloc).

6.1. Synthèse logique des descriptions SystemC

Dans cette section sera présentée la deuxième contribution qui traite le problème de la réalisation et la synthèse des composants matériels en définissant certaines règles ou un modèle de codage. La contribution cible une méthode pour éviter les problèmes qui peuvent apparaître pendant l'étape de génération automatique de la description RTL décrite en SystemC.

SystemC traite la non continuité entre la description comportementale de C/C++ et la description de HDL. Les concepteurs peuvent décrire leur conception dans SystemC et une fois qu'ils ont validé le concept dans SystemC, ils peuvent raffiner la description dans le niveau d'abstraction plus bas, comme par exemple une description de SystemC RTL. Ceci assure que les spécifications entre l'architecture de haut niveau et l'implémentation du système sont exécutables. Les vecteurs de test utilisés pour valider la conception dans un niveau plus élevé d'abstraction ont pu être réutilisés aux niveaux plus bas. Ces différentes descriptions en SystemC sont à la base des systèmes conçus à l'aide de l'environnement ROSES.

Dans l'objectif d'évaluer le flot de conception ROSES, il était développée une application inspirée de l'application VDSL [PET03]. Le flot ROSES était déjà évalué et vérifié avec des petits exemples facilement réalisables et les résultats obtenus ont été optimistes. Alors l'étape suivante était de concevoir une application avec une complexité plus grande proche de celle des

applications réelles. Le but était de conserver l'ensemble d'une architecture d'un modem VDSL développé par la société STMicroelectronics avec deux processeurs et d'un bloc de communication, et d'essayer de mimer le fonctionnement d'origine tout en constituant un véritable test pour le flot de conception ROSES. Le développement et les résultats de ce travail sont présentés dans [PAV02].

Une inattention a fait que les descriptions générées en SystemC, après l'étape de raffinement au niveau RTL par les outils du flot de conception ROSES, n'étaient pas toujours synthétisables. Nous avons effectué une analyse pour vérifier dans quel cas le code SystemC généré est synthétisable pour trouver quelles sont les constructions non synthétisables. On a choisi la synthèse logique pour effectuer cette analyse.

Il est très important de noter que la synthèse logique à partir de SystemC, telle que je la connais, passe toujours par une traduction intermédiaire de la description SystemC vers une description en VHDL ou Verilog. Ceci limite la sémantique qui peut être utilisée dans les descriptions synthétisables de ces circuits décrites en langage SystemC. Bien sur, il existe des travaux, qui recherchent des algorithmes permettant la synthèse logique directement à partir de SystemC, mais malheureusement pour le moment ce ne sont pas des approches effectives et applicables dans la pratique de la conception des systèmes monopuce.

Dans notre expérimentation l'étape de la synthèse logique a été réalisée en utilisant l'outil de synthèse Design Compiler de Synopsys, version 2003.06 qui supportait la synthèse directe à partir d'une description décrite en SystemC. Cet outil supportait la synthèse logique d'une description SystemC en format DB en utilisant la technologie Presto. La particularité de cette technologie était que l'étape d'élaboration pendant la synthèse passait par la transformation du langage SystemC en Verilog ou VHDL qui était utilisé par la suite pour la synthèse. L'inconvénient de cette technologie, comme on a mentionné précédemment, était qu'elle imposait plusieurs restrictions pour la description SystemC RTL synthétisable. La synthèse logique à partir de SystemC possédait un jeu des règles pour les constructions et les types des données qui sont synthétisables. Elles sont défini par [SYN02].

Les constructions SystemC énuméré dans le tableau suivant, montrent certaines des constructions non synthétisables.

Catégorie	Construction	Commentaire	Règle
Processus THREAD	SC_THREAD	Non synthétisable, utilisé seulement pour modéliser un vecteur de test mais non soutenu pour la synthèse.	Remplacer avec un processus SC_METHOD

Processus CTHREAD	SC_CTHREAD	Non synthétisable, utilisé seulement pour la simulation et pour modéliser des composants au niveau comportemental.	Remplacer avec un processus SC_METHOD
Conversation dynamique des types		Non synthétisable	Eviter. Choisissez bien les types de données
Variable globale		Non synthétisable	Remplacer avec des variables locales
Opérateur d'accès d'une structure	->	Non synthétisable, sauf pour l'instanciation d'un module	Remplacer avec l'opérateur point (.)
Pointeur	*	Non synthétisable, les pointeurs sont permis seulement dans des modules hiérarchiques pour instancier d'autres modules	Remplacer tous les autres pointeurs avec l'accès aux éléments ou à l'individu de rangée
Héritage		Non synthétisable	Créer un module indépendant en utilisant SC_MODULE

Dans cette section nous allons montrer quels sont les problèmes les plus fréquemment rencontrés pendant l'étape d'implémentation respectivement de la synthèse, de la description en SystemC de l'application VDSL, généré automatiquement par les outils du flot ROSES. Nous essayerons de donner une proposition du modèle de codage qui évitera ces problèmes. Un extrait de toutes les erreurs est montré dans l'exemple suivant :

```
...
GuardedRegister_HNDSHKo_VM2_VP2_HNDFIFOo.cpp

ERROR: "Invalid type conversion from `char' to `sc_lv<32>'(SCC-40)"
Warning: Conversion from `sc_lv' to `sc_bv' may result in loss of
information (SCC-136)
ERROR: "event-triggered if not supported in synthesis (SCC-112)"
Warning: Nonsignal member `status' is already accessed in `irq'
(SCC-410)
Warning: Nonsignal member `fifo' is already accessed in
`empile_depile' (SCC-410)
Information: Full-case automatically detected. (SCC-179)

GuardedRegister_HNDSHKo_VM2_VP2_HNDFIFOo.h

ERROR: The method that depends on both edge and non-edge expression
is not supported by synthesis not supported in synthesis (SCC-112)
...
```

Les erreurs rencontrées sont de type des conversions invalides des données et des déclarations invalides des méthodes de SystemC, conduite multiple des signaux et etc. Plus en détails :

SCC-40 (erreur) Conversion invalide de type % en type %.

Description : L'outil ne peut pas exécuter la conversion du type indiqué, tous les types des données ne sont pas soutenus pour la synthèse.

SCC-136 (avertissement) La conversion de type % à type % peut résulter dans une perte d'information.

Description : La conversion des types (en raison explicite ou implicite comme dépassement de paramètre, etc.) peut avoir comme conséquence la perte d'information. Ceci se produit quand la source a une largeur un peu plus grande que la destination.

SCC-112 (erreur) %s non soutenu pour la synthèse.

Description : L'outil de synthèse ne permet pas le code indiqué dans un processus synchrone « thread » d'être synthétisé.

SCC-410 (avertissement) Signal non membre de cette méthode est déjà modifié dans %.

Description : Cet avertissement indique que le signal non membre d'une méthode ne devraient pas être écrit à ou lu par plusieurs processus (méthodes) en raison de non déterminisme dans la simulation. Seulement des signaux membres de la méthode devraient être consultés par multiple processus.

SCC-179 (information) Détection automatique des cas.

Description : L'outil a automatiquement déterminé et complété tous les cas possibles dans une structure « switch ».

7. Proposition des technique de codage en SystemC

Cette section a pour but de montrer comment éviter certains des pièges communs, dans lesquels on peut tomber dans la conception du matériel utilisant le langage SystemC. Cette section, peut être, a un peu vieilli, vu le temps passé à partir de cette expérience (année 2003), mais quand même j'ai voulu la partager. Elle est contenue dans le présent manuscrit, pour donner à mes collègues un guide : comment concevoir des composants matériels, pour les bibliothèques du flot de conception ROSES, décrits en SystemC au niveau RTL.

Technique 1 : Découper la description d'un module en plusieurs méthodes simples.

Au cours des années, les ingénieurs de matériel développant des modules basés sur des langages HDL ont trouvé certains critères de découpage pour une conception plus efficace. Ces critères sont les suivants :

- ✓ Séparer le chemin de données et la logique de commande,
- ✓ Grouper les blocs matériels par rapport à la communication critique, la surface et la consommation de puissance,
- ✓ Découpage aux frontières de registres.

Les mêmes concepts sont applicables pour le code matériel décrit en SystemC.

Même après le découpage d'un module décrit dans une seule méthode en plusieurs petites méthodes, il est possible d'avoir des soucis pendant la phase de la synthèse due aux ressources de calcul disponibles. Chaque outil de synthèse a une limitation pratique sur la taille de module qu'il peut manipuler. Les tailles énormes d'un module peuvent exiger une grande quantité de mémoire et de puissance de calcul, qui peut seulement être présente dans une machine de calcul idéal. On croit que le découpage d'un module composé des méthodes multiples fournira un plus grand avantage pour l'utilisation de temps d'exécution et de mémoire. En outre, une fois que la conception est divisée, on peut appliquer différentes stratégies de synthèse pour obtenir le meilleur résultat.

Technique 2 : Il faut clairement séparer les parties combinatoires et séquentielles.

Cette règle fait partie d'un compromis entre la synthèse et la simulation. La séparation en logique séquentielle et combinatoire possède trois avantages :

(1) Elle aide à organiser la description du composant RTL (par exemple, penser à l'ingénieur qui doit lire cette description, et bien sûr on peut plus facilement décrire une conception pipelinée).

(2) Il existe une technique d'optimisation séquentielle dans tous les outils de synthèse, qui utilise le modèle de codage de l'élément séquentiel pour l'implémenter avec le meilleur composant séquentiel dans la bibliothèque de technologie ciblée. Quand on combine l'ensemble des descriptions de la logique séquentielle et combinatoire, l'outil de synthèse peut faire des confusions et ne pourrait pas identifier le type de composant séquentiel que l'on décrit.

(3) Pour des circuits avec un signal "reset" synchrone – le signal "reset" est facilement identifié et optimisé au cas où il y aurait des problèmes avec une description plus sophistiquée.

Technique 3 : Une seule méthode modifie un seul signal.

C'est un concept bien connu dans la conception des composants au niveau RTL que pas plus d'un processus peut forcer un signal particulier. Mais souvent les ingénieurs qui ont habitude d'écrire en C/C++, tentent de mettre l'initialisation de tous les signaux dans une méthode et d'implémenter la fonction logique dans des autres méthodes. Dans le monde de matériel, ceci crée des problèmes.

Exemple 1 : Modèle de codage - mauvais

```
sc_uint<4> count; // déclaration dans le fichier *.h
void reset() {
  if (reset.read() == 0x0)
  {
    count = 0x0;
    ... // d'autre code d'initialisation
  }
}
void count() {
  if (increment.read() == 0x0)
  { count++; }
  else
  { count--; }
}
```

Le code ci-dessus contient deux méthodes, les deux sensibles au front positif de l'horloge. Le concepteur veut initialiser toutes les variables globales dans une méthode simple « reset » de module. Ce qui suit est un exemple de code qui montre comment la fonction de « reset » et la fonctionnalité des signaux peuvent être implémentées en utilisant une méthode simple.

Exemple 2 : Modèle de codage - recommandé

```
sc_uint<4> count; // déclaration dans le fichier *.h

void count() {
  if (reset.read() == 0x0)
  { count = 0x0; }
  else {
    if (increment.read() == 0x0)
    { count++; }
    else
    { count--; }
  }
}
```

Technique 4 : Les types des données recommandés.

Il faut noter que souvent les types de données qui sont utilisés pour modéliser des signaux en SystemC, sont les types : *sc_uint, sc_int, et bool.* Le concept de SystemC fournit une représentation de deux valeurs pour ces types. Ceci signifie que si un signal est modélisé en utilisant ces types, il peut être conduit seulement avec deux valeurs contradictoires, alors il est possible que le simulateur ne détecte pas des problèmes des conduites multiples. Pour obtenir une

représentation plus précise des valeurs des signaux, il est préférable d'utiliser les types de données spécifiques tels que *sc_logic et sc_lv*. Ces types de données sont généralement recommandés pour modéliser les signaux à trois états pendant la simulation. La règle est que ces types des données ne sont pas synthétisés comme les types de données HDL *std_logic* et *std_logic_vector*. Ils sont traduits malheureusement toujours dans le type *unsigned.*

Technique 5 : Si on décrit un multiplexeur il faut implémenter toujours tous les cas pour éviter la génération des latches non désirés.

Comme en utilisant les langages Verilog et VHDL, si on ne fait pas attention au codage des modules, on peut insérer dans le circuit beaucoup de latches ou des registres non désirés.

Exemple 3 : Latches – mauvais modèle de codage

```
sc_out<bool > output; // déclaration dans le fichier *.h

void latch() {
  if (condition.read() == 0x1)
  { output.write(0x1); }
  else
  {
    // rien pour output, il gardera sa valeur
  }
}
```

Le code décrit comme ça sera implémenté comme un latch. La solution la plus simple est d'ajouter "output.write(0x0)" avant "if", ou à l'intérieur de "else".

Exemple 4 : Latches - modèle de codage recommandé

```
sc_out<bool > output; // déclaration dans le fichier *.h

void non_latch () {
  if (condition.read() == 0x1)
  { output.write(0x1); }
  else
  { output.write(0x0); }
}
```

Technique 6 : Il faut réinitialiser localement la valeur d'un variable intermédiaire pour éviter la génération des registres non désirés

Si on ne fait pas attention, tout en écrivant un code complètement synthétisable en SystemC, on peut générer aussi des registres inutiles. Ceci se produit quand une variable intermédiaire est utilisée dans une ou plusieurs branches d'une construction « if-elseif-else ». Ce comportement est semblable à ce qui est vu dans Verilog et VHDL. Pour ces parties des branches où la valeur de la variable intermédiaire n'est pas définie, l'outil de synthèse interprète que la vieille valeur doit être maintenue, et il génère un registre supplémentaire. Voici un exemple.

Exemple 5 : Registres supplémentaires – mauvais modèle de codage

```
// déclaration dans le fichier *.h
// bool add;
// sc_unit<8> a,b,IntReg,Reg;

void register() {
  if (reset.read() == 0x0) {
  { Reg = 0x0; }
  else {
   if (add)
   {
     IntReg = a;
     Reg = IntReg + b;
   }
   else
   { Reg = a - b; }
  }
}
```

Exemple 6 : Registre supplémentaire – modèle de codage recommandé

```
// déclaration dans le fichier *.h
// bool add;
// sc_unit<8> a,b,c,IntReg,Reg;

void no_regsiter() {
  IntReg = 0x0; // réinitialization locale
  if (reset.read() == 0x0)
  { Reg = 0x0; }
  else {
    if (add)
    {
      IntReg = a;
      Reg = IntReg + b;
    }
    else
    { Reg = a - b; }
  }
}
```

Une remarque recommandée est de vérifier le fichier « *.log » de la synthèse pour trouver s'il existe des registres supplémentaires. Ce fichier montre les détails des registres générés par l'outil de conception (comme par exemple Design Compiler), quel est le type de registre (flip-flop/latches), la largeur du registre et etc. Mais cette tâche est délicate.

Technique 7 : La conversation des types des données

La conversation dynamique des types des données provoque des problèmes. Le « cast » logiciel n'est pas approprié au sens matériel. Par exemple la conversation de l'entier sur 8 bits en vecteurs sur 8 bits provoque une erreur de la synthèse. Alors tout simplement elle est interdite. C'est pourquoi il faut bien déterminer dès le début les types des données qu'on va utiliser dans la conception de la partie matérielle.

Exemple 7 : La conversation des types des données – interdit

```
sc_module(toto) {
...
sc_uint<8> xyz;
bool abc;
...
void titi() {
...
abc = or_reduce((sc_bv<8>)xyz); // Interdit
...
}
}
```

Technique 8 : Les méthodes et leurs listes de sensitivité

Les outils de synthèse ne soutiennent pas actuellement la synthèse d'autres méthodes SystemC que la méthode « sc_method ». Il existe aussi certaines limitations dans la déclaration de leurs listes de sensitivité. Ces limitations provoquent l'erreur suivante pendant l'étape de la simulation :

```
ERROR: The method that depends on both edge and non-edge expression
is not supported by synthesis not supported in synthesis (SCC-112)
```

Ainsi, la déclaration suivante est illégale pour la synthèse de matériel :

Exemple 8 : Méthodes – mauvais modèle de codage

```
void seq() {
  if (reset.read())
    { SelReg = 0x0; }
  else if (clock.read())
  {
    if (ready.read() == 1)
      { SelReg = Sel; }
  }
}
...
SC_CTOR(smi) {
  SC_METHOD(seq);
    sensitive << reset;
    sensitive_pos << clock;
...
```

La solution de ce problème est démontrée dans l'exemple suivant :

Exemple 9 : Méthodes - modèle de codage recommandé

```
...
void seq() {
  if (reset.read() == 0)
  { SelReg = 0x0; }
  else if (clock.read() == 1)
  {
    if (ready.read() == 1)
      { SelReg = Sel; }
  }
}
```

```
...
SC_CTOR(smi) {
  SC_METHOD(seq);
    sensitive << reset << clock;
...
```

Une remarque la primitive « event() » n'est pas non plus synthétisable.

```
ERROR: "event-triggered if not supported in synthesis (SCC-112)"
```

Alors il faut éviter l'utilisation des constructions de genre :

```
void fsm() {
  if (reset.read()== active)
  { current_state = idle_state; }
  else if(clock.event()) {
  { current_state = next_state; }
}
```

8. Types de données soutenus

Pour tous les concepteurs de matériel qui raisonnent comme s'ils écrivaient un programme logiciel, on lui recommande d'examiner le sous-ensemble synthétisable de types de données de SystemC. Sans se rendre compte du sous-ensemble synthétisable, on va développer un code, qui peut être seulement simulable. Il est recommandé de regarder, quels sont les types des données synthétisables pour l'outil de conception utilisé. En général, les règles pour utiliser les différents types des données en SystemC, sont les suivantes :

- ✓ Pour une variable de type bit unique, utiliser le type bool de C++.
- ✓ Pour des variables avec une largeur de 64 bits ou de moins, employer le type de données de sc_int ou de sc_uint.
- ✓ Pour des variables plus en grande partie que 64 bits, utiliser le sc_bigint ou le sc_biguint.
- ✓ Employer sc_logic ou le sc_lv seulement quand il faut modéliser des signaux à trois états pour un bus. Quand ces types sont utilisés, il faut éviter la comparaison avec des valeurs état inconnue « X » et haute impédance « Z », parce que de telles comparaisons ne sont pas synthétisables.
- ✓ Utiliser des types de données propres de C++ avec prudence, car ils dépendent souvent de la plateforme utilisée pour la simulation.
- ✓ Les outils de synthèse de SystemC ne soutiennent pas quelques types de données :
- ✓ les types à virgule flottante,
- ✓ les types à virgule fixe sc_fixed, sc_ufixed, sc_fix, et sc_ufix,
- ✓ les types tels que des pointeurs, car les descriptions en SystemC RTL ne suppose pas l'existence d'une mémoire et d'espace d'adressage,
- ✓ les types classe tels que le type fichier.

Enfin, c'est nécessaire de redire que cette expérience date depuis l'année 2003, en espérant que les outils de synthèse à partir de SystemC ont progressé et ont amélioré certains aspects des problèmes traités dans la dernière partie de ce chapitre.

9. Conclusion

Dans ce chapitre, nous avons étudié et examiné les différents modèles et méthodes concernant la conception des systèmes monopuce à partir du niveau RTL. Nous avons d'abord présenté les concepts généraux dans la modélisation des systèmes électroniques, c'est-à-dire la description d'entrée au niveau RTL, la description de sortie au niveau logique et le flot de conception classique pour la conception physique basé autour du langage VHDL. Nous avons ensuite abordé la méthodologie et le processus de covérification par simulation des systèmes digitaux. Nous avons également étudié les particularités et la spécificité des modèles et des interfaces de simulation. Enfin nous avons présenté l'un des objectifs de cette thèse et également une des contributions qui traite le problème de génération automatique des descriptions au niveau RTL, en formalisant la réalisation et la synthèse des composants matériels décrits en SystemC en définissant certaines règles de codage. Cette contribution vise un modèle de codage pour éviter les problèmes qui peuvent apparaître pendant l'étape de la synthèse pour les descriptions décrites en SystemC RTL générés automatiquement.

Chapitre 5

EXPERIMENTATION : L'ENCODEUR DIVX

Ce chapitre présentera un exemple d'utilisation des techniques décrites dans les deux chapitres précédents : la méthode d'intégration de matériel et logiciel à partir d'un modèle au niveau de bus fonctionnel et l'implémentation et la réalisation physique des composants matériels dans le processus de prototypage d'une application de codage de vidéo en temps réel.

1. Introduction

Nous présentons une étude de cas de conception d'un environnement d'exécution et de vérification de matériel pour un système monopuce multiprocesseur. Le problème de conception a été concentré sur le développement d'une plateforme appropriée de vérification au niveau RTL pour fusionner le logiciel et le matériel de l'encodeur DivX dans un prototype correspondant aux conditions de modèle et de spécifications de système de haut niveau. Le chapitre décrit la conception de l'architecture du matériel autour du bus AMBA en utilisant la méthodologie d'intégration de composants d'IP et de l'environnement approprié de vérification en utilisant un modèle de simulation du processeur ARM946E-S. Nous allons présenter aussi les résultats de la synthèse et de l'estimation de la puissance consommée.

2. Description de l'application DivX

Au cours des dernières années dans le groupe SLS du laboratoire TIMA, nous avons développé le prototype d'une application de codage de vidéo en temps réel basé sur l'implémentation OpenDivX [MAY03] de la norme MPEG-4 [KOE02]. L'architecture de cette application est le résultat de plusieurs travaux de recherche dans le groupe SLS. Cette application a servi à expérimenter pendant le développement des concepts et des outils de l'environnement ROSES du groupe SLS.

2.1. Le standard MPEG et l'implémentation OpenDivX

MPEG-4 (ISO/IEC 14496), introduit en 1998, est une norme de codage d'objets audiovisuels spécifiée par le Moving Picture Experts Group.

MPEG-4 est d'abord conçu pour gérer le contenu de scènes comprenant un ou plusieurs objets audio/vidéo. Contrairement à MPEG-2 qui visait uniquement des usages liés à la télévision numérique (diffusion DVB et DVD), les usages de MPEG-4 englobent toutes les nouvelles applications multimédias comme le téléchargement et le « streaming » sur Internet, le multimédia sur les téléphones mobiles, la radio numérique, les jeux vidéo, la télévision et les supports haute définition.

MPEG-4 a développé de nouveaux encodeurs audio et vidéo et enrichi les contenus multimédia, en ajoutant de nouvelles technologies comme VRML (Virtual Reality Modeling Language), support pour des présentations 3D, des fichiers composites orientés objet (incluant des objets audio, vidéo et VRML), le support pour la gestion des droits numériques et plusieurs types d'interactivité.

MPEG-4 se décompose en une suite de normes, les parties, qui spécifient un type de codage particulier. Dans chaque partie plusieurs profils (collection d'algorithmes) et niveaux (contraintes quantitatives) sont définis. Un consortium industriel désirant utiliser MPEG-4 choisit une ou plusieurs parties de la norme et, pour chaque partie, il peut sélectionner un ou plusieurs profils et niveaux correspondant à ses besoins.

Le DivX est une technologie pour compresser un fichier vidéo sans réduire la qualité visuelle. Cette technologie est basée sur la norme de compression MPEG-4. Elle permet de compresser un fichier au format MPEG-4 jusqu'à 10 % de sa taille d'origine. Malheureusement, l'encodeur est très complexe et l'implémentation temps réel devient extrêmement difficile. La figure 5.1 montre le schéma de l'encodeur DivX.

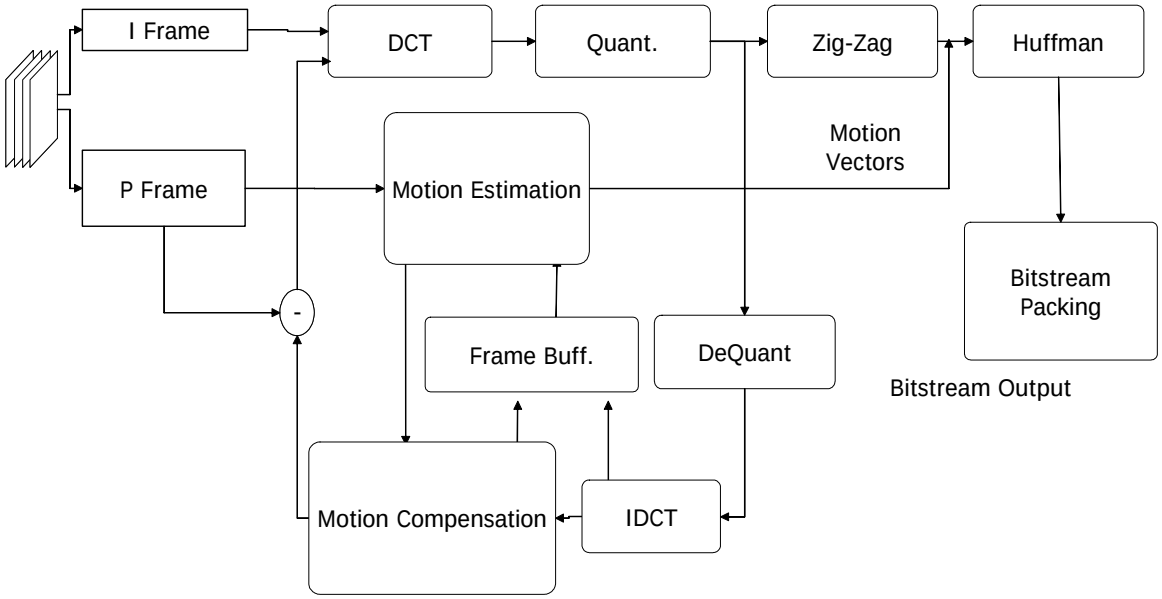


Figure 5.1 : Le schéma fonctionnel de l'encodeur DivX.

L'encodeur DivX accepte en entrée une séquence d'images au format YUV 4 : 1 : 1. Ce format présente l'image en utilisant trois matrices pour décrire luminance et chrominances. La matrice Y décrit la luminance, et les matrices U et V présentent respectivement la chrominance blanc/rouge et la chrominance blanc-bleu. La technologie exploite les réductions des redondances spatiale et temporelle. La redondance temporelle est réduite en décrivant une image par ses différences seulement par rapport à l'image précédente. L'image compressée de cette façon est appelée image P. La redondance spatiale est réduite en utilisant la compression JPEG.

Le DivX est une application très complexe. La version gratuite de cette application est appelée OpenDivX. L'OpenDivX peut être compilé en utilisant un compilateur standard de C et exécuté sur PC. Cette application contient environ 10000 lignes de code C. Le temps d'exécution de ce programme prend environ une seconde pour une séquence de 20 petites images en résolution QCIF (176 x 144 pixels) sur un processeur Athlon 4 à 1200 MHz. C'est très loin des besoins de la

plupart des formats vidéo qui nécessitent 30 images par seconde avec des formats beaucoup plus grands (CCIR 720 x 576, CIF 352 x 288, HDTV 1280 x 720 – 60 images par seconde).

2.2. Les objectifs pour le prototypage de l'encodeur MPEG4

Le groupe SLS du laboratoire TIMA a voulu développer une application pour pouvoir valider et tester nos concepts de conception des systèmes monopuce multiprocesseurs. C'est pourquoi nous avons choisi de concevoir un système multiprocesseur monopuce pour l'encodage de vidéo en temps réel en employant le standard de codage OpenDivX. Comme nous avons vu dans le paragraphe précèdent, la réalisation d'une application de codage de vidéo est très complexe et nécessite une puissante ressource de calcul. Alors pour pouvoir implémenter cette application dans un système monopuce multiprocesseur, il était nécessaire de paralléliser le code C de l'algorithme et son exécution. L'algorithme de codage était découpé et partagé, par les travaux de [WAS04], [SAS04] et [PAV04], entre plusieurs processeurs afin de diminuer le temps d'exécution pour atteindre l'encodage en temps réel avec un système embarqué.

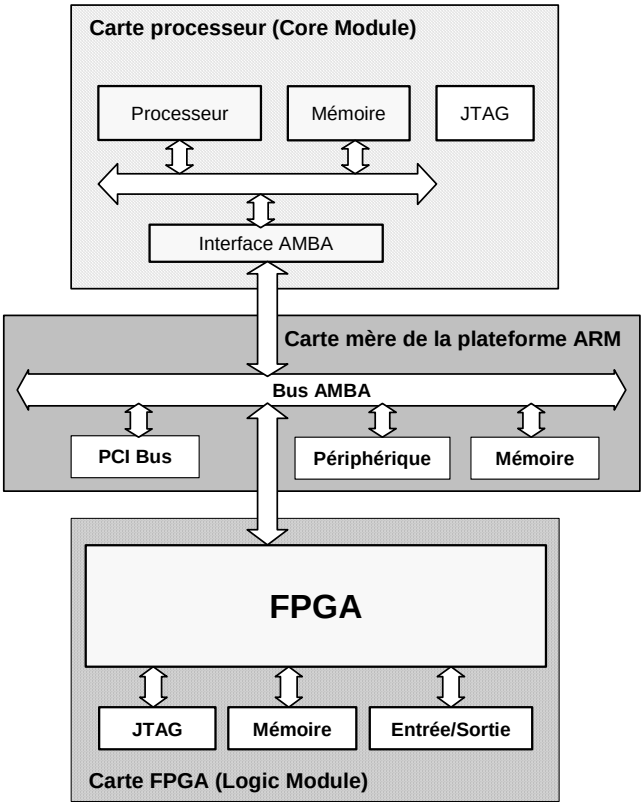


Figure 5.2 : La plateforme de prototypage rapide ARM Integrator.

Avant de réaliser cette application sur une puce de silicium, les travaux de thèse d'Arif Sasongko [SAS04] ont développé un prototype sur une plateforme ARM Integrator. La plateforme ARM Integrator est une plateforme matérielle conçue pour le développement des applications et pour le prototypage des composants matériels autour des processeurs ARM et du bus AMBA.

Cette plateforme contient trois types de carte : une carte mère (nommée « ARM Integrator AP »), une carte processeur (nommée « core module »), et une carte FPGA (nommée « logic module »). La figure 5.2 montre une version simplifiée de cette plateforme avec trois cartes processeurs et deux cartes FPGA.

Les objectifs de ces travaux de thèse étaient : (1) de vérifier le code parallélisé, (2) de vérifier le système d'exploitation pour être sûr qu'il marche sur une configuration multiprocesseur, et (3) d'insérer un mécanisme de communication.

Malheureusement le prototypage complet de l'application sur cette plateforme est devenue une tâche très difficile et il n'était pas possible de valider l'architecture entière de l'encodeur. En effet, la conclusion de la réalisation de l'application sur la plateforme ARM Integrator était que le prototype n'avait pas la même performance que le système monopuce final. Cela était dû à la fréquence d'horloge du processeur sur la plateforme qui est beaucoup plus lente (20 MHz) par rapport à la fréquence souhaitée (60 MHz), et aussi à l'architecture du réseau de communication sur la plateforme (un bus AMBA) qui était fixée avec une bande passante très limitée.

C'est pourquoi, mes travaux de thèse ont eu pour but de procéder directement avec le prototypage physique du système. Nous avons également pensé à valider l'architecture de l'application, à vérifier le système d'exploitation et de valider la communication entre les différents modules, en utilisant des environnements de cosimulation à chaque niveau d'abstraction employé pendant le processus de prototypage physique de l'application.

2.3. La spécification et l'architecture de l'application

Comme mentionné dans le paragraphe précédemment, les travaux de thèse de Wassim Youssef et Marius Bonaciu du groupe SLS ont eu pour but de partitionner le code initiale de l'algorithme DivX entre plusieurs processeurs. Le code de l'encodeur DivX a été partitionné en deux modules avec une configuration maître et esclave (voir figure 5.3). Le maître effectue le processus du calcul de plus haut niveau. L'esclave effectue le calcul du vecteur de mouvement et la compensation du mouvement. Un protocole MPI (Message Passing Interface) [MPI95] était mis en place par les travaux de thèse de Yannick Paviot comme une primitive de communication entre les deux modules. Le protocole MPI est une interface standard pour la communication entre plusieurs tâches sur une machine parallèle. Dans notre application, il était implémenté que les deux primitives : *MPI_send* et *MPI_receive*. Ces deux fonctions permettaient de réaliser des communications bloquantes entre les tâches en utilisant des FIFO. Plus des détails pour cette réalisation peuvent être trouvés dans [PAV04].

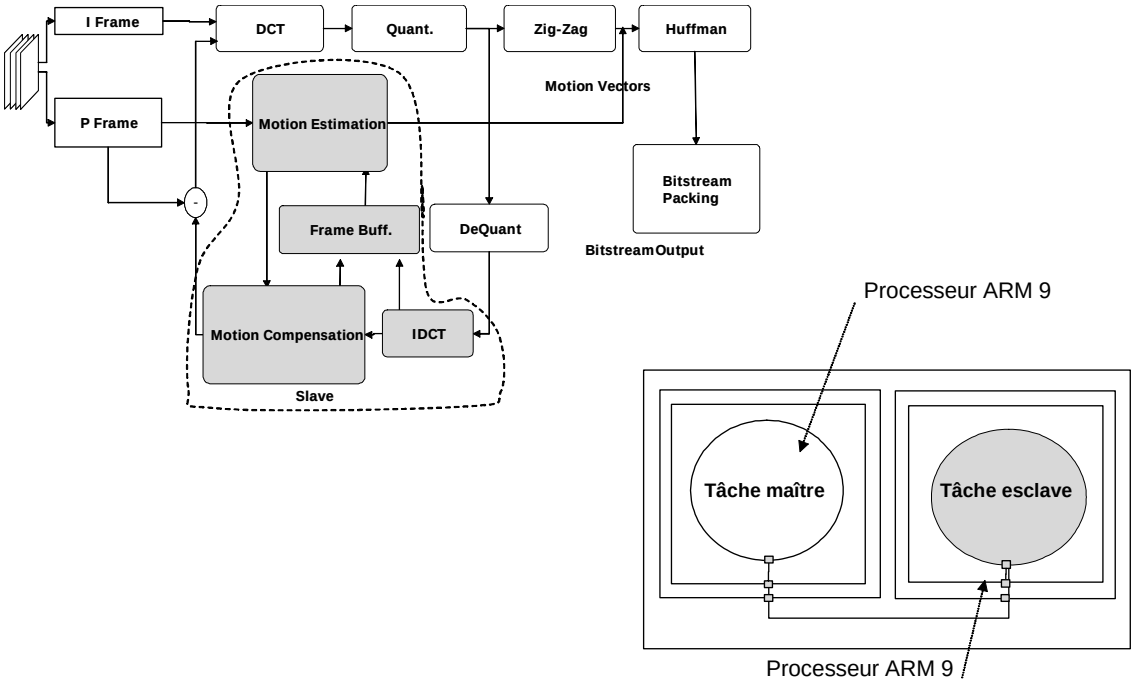


Figure 5.3 : La spécification de départ de l'application DivX

2.4. Les contraintes de l'application

Nous avons voulu également créer une application pour les domaines des applications portables. Cela a imposé que nous avons fixé la fréquence d'horloge du système à 60 MHz, pour diminuer et concevoir une application de basse consommation. Comme nous avons déjà eu l'expérience avec les processeurs ARM et le bus AMBA sur la plateforme ARM Integrator, nous avons continué à travailler avec ces processeurs. Il est nécessaire de mentionner que nous avons fixé la fréquence d'horloge du processeur ARM946E-S qui était intégré finalement dans le prototype finale de l'encodeur MPEG4 tandis que ce processeur permet de travailler à une fréquence beaucoup plus élevé (cf. page 128).

3. L'architecture de l'application aux différents niveaux d'abstraction

Cette section présente l'architecture du système monopuce multiprocesseur pour codage vidéo conçu dans le groupe de SLS au laboratoire TIMA. Nous allons décrire les différentes représentations de l'architecture utilisée à chaque étape de la conception, ainsi que certaines particularités des composants implémentant son architecture.

3.1. L'architecture de l'application DivX

L'architecture de l'application DivX a été initialement développée et validée au niveau système en utilisant l'environnement de conception ROSES. L'environnement ROSES représente une méthodologie de conception et de vérification de matériel et de logiciel. Il est capable de prendre comme entrée après le découpage et l'exploration de l'architecture des descriptions en SystemC (dans notre cas VADeL, une extension de langage SystemC) comprenant des paramètres d'exécution et de raffiner cette description au niveau système à une description au niveau de bus fonctionnel. La figure 5.4 montre le découpage de l'architecture de l'application DivX après l'exploration de l'architecture et la figure 5.5 montre la description obtenue après le raffinement automatique de matériel et de logiciel en utilisant l'environnement ROSES.

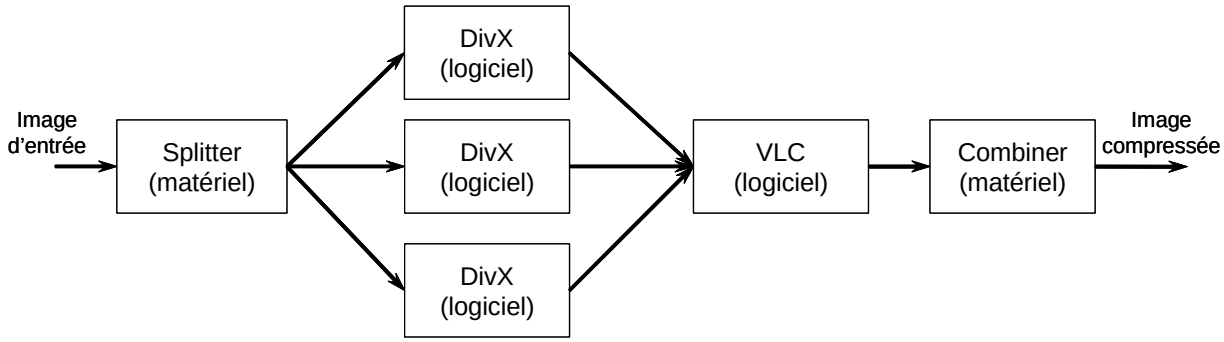


Figure 5.4 : L'architecture de l'encodeur DivX après le découpage de matériel et de logiciel.

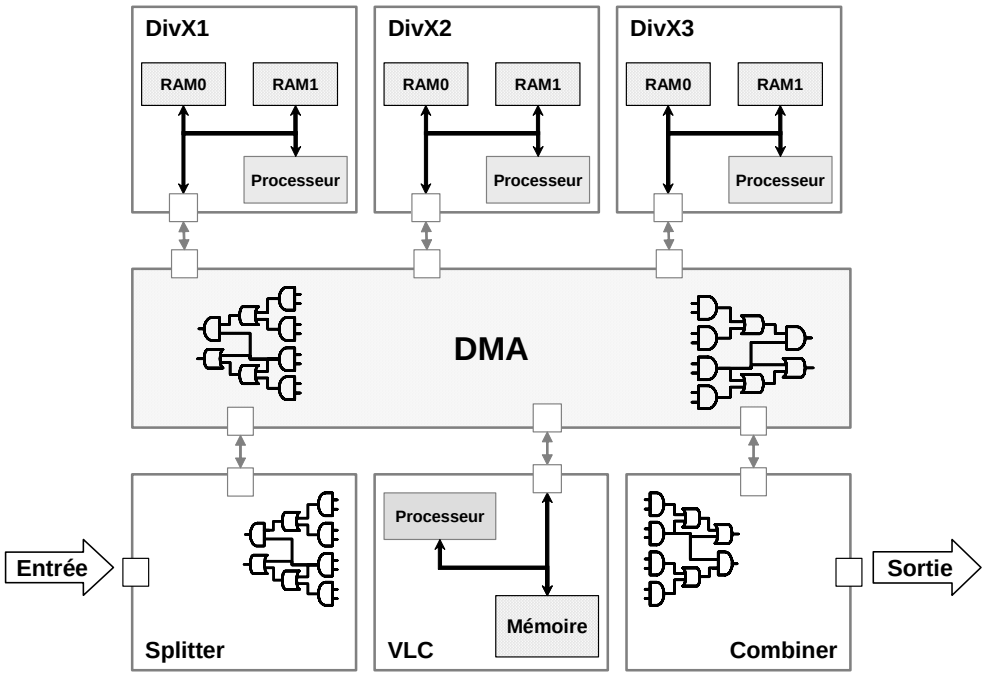


Figure 5.5 : L'architecture de l'encodeur DivX après le raffinement automatique.

L'architecture de l'application DivX est composée de trois sous-systèmes d'entrée, nommées DivX, et un sous-système de sortie nommée VLC (Variable Length Coder). Nous avons

implémenté chacune des quatre tâches dans un des sous-systèmes de l'architecture multiprocesseurs. La tâche « maître » est implémentée sur un processeur ARM9 dans le sous-système VLC, réalisant le codeur d'entropie. Les autres trois des tâches sont implémentées sur le processeur ARM9 des sous-systèmes DivX, réalisant l'évaluation et la compensation de mouvement, de la Transformée Discrète en Cosinus (DCT) et de la quantification. Un contrôleur d'accès direct à la mémoire (DMA), est utilisé pour interconnecter les quatre sous-systèmes pour établir la communication entre tous les modules des sous-systèmes et aussi des modules d'entrée et de sortie.

Le flux fonctionnel de l'encodeur DivX est comme suit :

Chaque image vidéo à l'entrée est divisée en trois parties par l'interface d'entrée, nommé « Splitter ». Chaque partie de l'image découpée est renvoyée à son tour vers l'un des trois sous-systèmes de DivX : DivX1, DivX2 et DivX3. Les sous-systèmes DivX traitent l'image et préparent les données pour la compression. Une fois que les données de chaque sous-systèmes DivX sont préparées, elles sont transférées au sous-système de VLC, où la compression de l'image entière est finalisée et les données compressées sont envoyées à l'interface de sortie, nommée « Combiner » pour ajuster certaines paramètres de compression et les transférer vers une unité de stockage.

Tous les transferts de données entre les sous-systèmes sont effectués par un contrôleur DMA spécifique. Ce contrôleur DMA est particulièrement conçu pour l'application, qui représente une implémentation complètement différente par rapport à des contrôleurs d'accès direct à la mémoire standards décrits dans la littérature. De même façon, tous les blocs de matériel hors des sous-systèmes sont aussi conçus spécifiquement pour l'application dans le groupe SLS du laboratoire de TIMA. Chacun des ces blocs matériels possède plusieurs réalisations : ils ont souvent été conçus par mes collègues sous préoccupation d'être implémentés sous forme de circuit. L'objectif initial n'était que la simulation du fonctionnement. C'est pourquoi une partie de mes travaux était consacré à la conception de l'implémentation physique de ces blocs, tout en respectant les fonctionnalités décrites par mes collègues.

3.2. Le prototype virtuel de l'application

Comme nous avons déjà décrit le prototype virtuel représentait la description initiale au niveau de bus fonctionnel pour les travaux de cette thèse. Cette description était composée d'une partie logicielle contenant l'application, le système d'exploitation et la couche d'abstraction de matériel générique, et d'une partie matérielle représentant tous les composants de calcul, de mémorisation et de communication du système. Le prototype virtuel de l'encodeur DivX, était initialement développé en utilisant un environnement de co-simulation basé sur plusieurs

simulateurs différents comme celui décrit dans le chapitre 3. Nous avons utilisé un simulateur ISS pour simuler le processeur (ARMulator) et exécuter la partie logicielle et un simulateur SystemC pour simuler le reste de la partie matérielle.

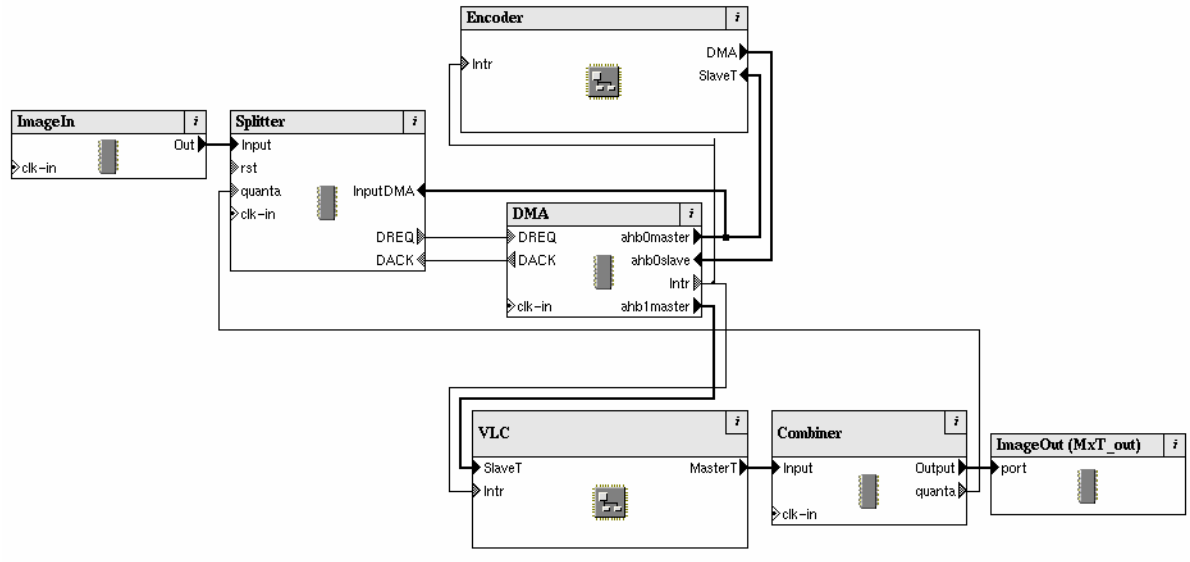


Figure 5.6 : Le prototype virtuel de l'encodeur DivX.

Dans une étude comparative, nous avons migré vers l'environnement du simulateur MaxSim, qui était décrit aussi précédemment dans le chapitre 3. La figure 5.6 présente le prototype virtuel de l'encodeur DivX développé en utilisant l'environnement MaxSim et les composants disponibles dans la bibliothèque telle que le processeur ARM9 et les mémoires TCM, le bus AMBA, le contrôleur d'interruption et le « timer ». Cette bibliothèque a été élargie avec des composants spécifiques à l'application comme le « Splitter », le « Combiner » et le contrôleur DMA.

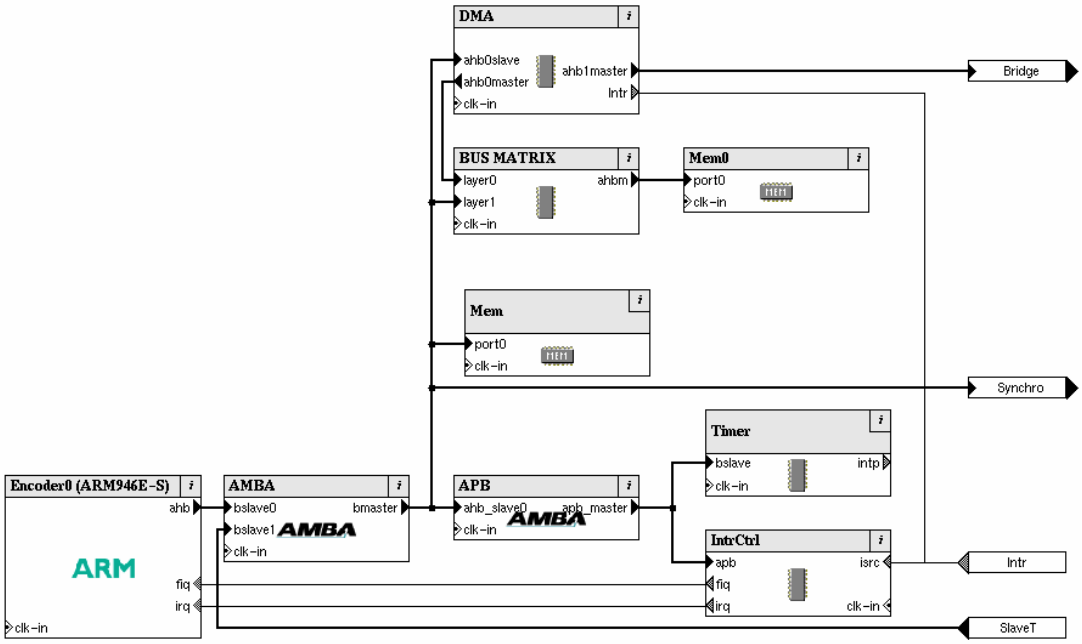


Figure 5.7 : Le sous-système DivX de l'encodeur MPEG4.

La figure 5.7 présente plus en détail l'architecture locale de l'un des sous-systèmes du prototype virtuel de l'encodeur DivX dans MaxSim. La figure montre l'implémentation du sous-système DivX. Comme nous pouvons voir, ce sous-système se compose d'une description du processeur ARM9E-S, d'un contrôleur d'interruption, d'un « timer », d'un contrôleur DMA, des mémoires et d'une matrice de bus « BusMatrix », responsable pour fournir l'accès parallèle aux deux bancs de la mémoire vidéo. Chaque composant est interconnecté via un modèle fonctionnel du bus AMBA. Tous les composants constituants les sous-systèmes de l'encodeur DivX dans le cette représentation sont modélisé au cycle près, pour cela chacun des composants sur les figures 5.6 et 5.7 possède un port « clk-in » qui est lié implicitement avec l'horloge du système.

Le prototype virtuel a été utilisé pour valider le logiciel et l'architecture de l'application. Il a servi aussi pour le développement du logiciel dépendant du matériel, tel que les pilotes spécifiques au contrôleur d'interruption et au contrôleur DMA. Le prototype virtuel a été employé également pour l'analyse de la performance du logiciel et de la communication, comme représenté sur la figure 5.8.

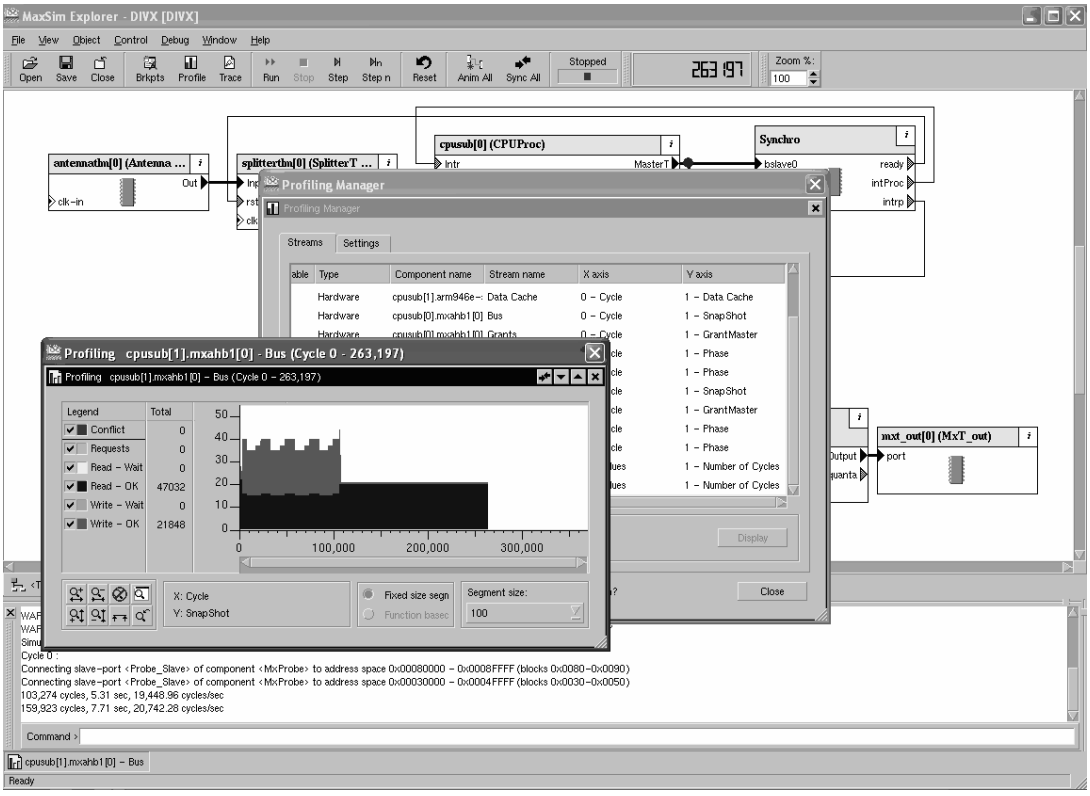


Figure 5.8 : L'analyse de la communication sur le bus du processeur du sous-système DivX.

La figure 5.8 présente brièvement l'analyse de la communication pour l'un des sous-systèmes de l'encodeur DivX, profilant les conflits, les requêtes et les lectures et les écritures terminées avec succès sur le bus pour une image de la séquence vidéo.

3.3. Les facteurs affectant la performance

Dans ce paragraphe, nous allons présenter certaines des facteurs qui affectent la performance d'un système embarqué et en particulier un système monopuce multiprocesseur. C'est ici l'endroit pour mentionner ces facteurs, car ils sont les principaux critères utilisés pour concevoir l'architecture de l'encodeur DivX au niveau RTL et pour choisir les composants implémentant ce système. Ce paragraphe a pour but de justifier les choix faits par le concepteur pendant le développement de ce système. On peut noter aussi que la majorité de ces aspects ne peuvent pas être pris en compte au niveau de bus fonctionnel.

3.3.1. Les effets internes affectant la performance

Les caches : Les caches d'instructions et de données ont un rôle important sur la performance de l'exécution du programme. Les caches permettent à une partie du programme ou des données d'être stockée dans une mémoire plus petite et plus rapide. Si les données ou les instructions sont employées fréquemment ceci va accélérer la performance. Si les données ou le code sont employés rarement, ceci peut réellement ralentir l'exécution.

Un ISS peut modéliser les caches s'il assigne la mémoire pour le cache et maintient le contenu de cache, utilisant le même algorithme pour le remplissage des lignes de cache. Pour certains processeurs ARM, la taille du cache est fixe, pour d'autres elle peut être définie par le fabriquant de système.

Accès mémoire ou accès au cache : La première complication est que, même pour un processeur RISC comme les processeurs ARM, pas toutes les instructions ont le même temps d'exécution. Un des facteurs qui peuvent affecter la vitesse d'exécution est le nombre de cycles du bus produits par une instruction. Parce que certaines instructions doivent simplement être recherchées dans la mémoire ou dans le cache. Tandis que d'autres instructions produisent un accès des données sur le bus, aussi bien que le « fetch » des instructions recherchées. D'autres instructions peuvent produire des accès mémoire multiples – considérer les instructions de LDM ou de STM du jeu d'instruction ARM, qui peut causer jusqu'à 17 accès de mémoire. S'il n'y a aucun cache, et si les accès mémoire sont lents, les accès mémoire seront le goulot d'étranglement de l'exécution du logiciel et peuvent être utilisés pour obtenir une évaluation grossière à la sortie du système. Si les caches sont mis en place, l'exécution d'instruction ou les accès de mémoire peut être le facteur de détermination de la performance du système. La plupart des simulateurs ISS tiendront compte de cette complication.

Accès mémoire au niveau physique : Toutes les instructions d'un processeur ne sont pas égales. De même façon, pas tous les accès mémoire sont identiques. Les systèmes embarqués

doivent souvent équilibrer entre la performance et la basse consommation, respectivement le bas coût. Ceci signifie que les systèmes souvent se composent de différents types de mémoires avec des vitesses différentes. Pour compliquer encore le sujet, les différents accès vers une mémoire peuvent avoir des temps d'exécution différents dans le cas du ARM. On doit tenir compte aussi pour les cycles « *burst* » et les cycles « *séquentiels* ». Les cycles « *burst* » qui transfèrent des mots multiples à partir de la mémoire au processeur, souvent à remplir une ligne du cache, mais ainsi parfois pour une lecture ou écriture multiple des registres. Les cycles « *séquentiels* » sont des accès mémoire où des mémoires adjacentes sont consultées. Quelques protocoles de bus permettent un transfert plus rapide des données qui sont mises en place en tant qu'accès séquentiels. Malheureusement cette complication n'est pas prise en compte ni par les simulateurs ISS ni par les descriptions du processeur de haut niveau.

La largeur de la mémoire : Un autre problème qui affecte la performance du code est la largeur des blocs de mémoires. Pour différentes raisons, une région de mémoire ne peut pas couvrir la pleine largeur du bus de donnée vers la mémoire du processeur – dans le cas des processeurs ARM, le bus de donnée vers la mémoire est 32 bits. Si la largeur du bus de donnée de la mémoire est 8 bits, elle prendra quatre cycles pour obtenir une valeur de 32 bits sur le bus de donnée du processeur. Les quatre cycles ne seront pas des transactions complètes sur le bus et il ne sera pas ainsi quatre fois plus lent, mais ce sera un peu plus lent qu'un accès de 32 bits. D'après la littérature, l'accès d'un bus de 32 bits à partir d'une mémoire de 8 bits est environ 3 fois plus lent que l'accès d'une mémoire de 32 bits. C'est pourquoi très souvent on intègre quatre blocs de mémoire de 8 bits en parallèle pour couvrir la larguer d'un bus de 32 bits. Ce schéma d'intégration a ses avantages car le temps nécessaire pour rechercher un mot dans la mémoire diminue. Ce fait est dû à la baisse de la latence grâce aux plus petites mémoires nécessaires à implémenter.

Les mémoires TCM (Tightly Coupled Memory) : Plusieurs processeurs ARM ont une mémoire TCM. C'est une mémoire rapide qui fait partie du cœur du processeur, qui aura les caractéristiques d'exécution semblables aux caches, mais elle est située à une adresse fixe. La taille de ces mémoires peut être définie par le concepteur de système, ou elles peuvent être omises.

La vitesse de bus : Dans les systèmes basés sur les processeurs ARM, comme quelques autres processeurs, la fréquence d'horloge du processeur peut être différente de celle du bus du système. Des caches seront accédés à la vitesse du processeur, ainsi que les mémoires TCM, mais la mémoire principale externe sera mis en place avec la fréquence d'horloge plus lente du bus. La différence entre ces deux fréquences aura un impact sur la performance totale du système, et influencera de manière significative l'importance d'équilibrer la taille des caches avec l'application.

La mémoire-tampon (write buffer) : Quelques processeurs de ARM ont un « *write buffer* » ou *mémoire-tampon*, qui permet à des données étant écrites de nouveau dans la mémoire d'être gardées jusqu'à ce que le bus devienne disponible. Ceci permet une exécution plus rapide du système global, parce qu'il permettra au processeur de continuer à travailler même si le bus est occupé et une opération d'écriture d'être accomplie. Normalement, le processeur devrait attendre que le cycle d'écriture soit accompli avant de continuer avec l'opération. Le « *write buffer* » du processeur ARM946E-S permet de stocker jusqu'à 16 opérations d'écriture avant que le processeur attende l'accès sur le bus. Le « *write buffer* » est particulièrement efficace si l'horloge du bus fonctionne à une vitesse plus lente que l'horloge de processeur.

3.3.2. Les effets externes affectant la performance

La capacité d'un dispositif externe à accepter des données pourra être un problème sur le comportement du système. Pour notre système, la partie des données vidéo sont fournies à la mémoire par une autre partie du système qui n'est pas affectée par l'exécution du processeur. Puisque le transfert des données est plus rapide que le traitement des données par le processeur cela n'est pas significatif pour l'analyse. Cependant, ceci devra être considéré car notre système est dépendant de la vitesse de la mémoire externe. Il est possible de modéliser ceci d'une manière quelconque avec un ISS mais modéliser ceci dans HDL est beaucoup plus facile – ceci est particulièrement vrai si le matériel est impliqué dans le traitement des données avant qu'elles soient livrées au processeur.

L'arbitrage du bus pour DMA et d'autres maîtres du bus : En plus des facteurs qui détermineront combien de temps un cycle du bus prendra pour s'accomplir, s'il y a d'autres maîtres du bus dans le système, le temps d'acquisition du bus devra être pris en considération. Le bus est une ressource partagée entre les divers constituants qui peuvent lancer des transactions sur le bus. Un processeur est toujours un maître du bus, car il devra chercher des instructions ou lire ou écrire des mémoires. Il peut y avoir d'autres maîtres du bus tels que des contrôleurs DMA, ou d'autres processeurs DSP. Quand il y a plus d'un maître du bus dans le système, il aura besoin pour demander l'accès au bus d'un arbitre du bus. La nature de l'arbitre et de l'algorithme qu'il emploie pour déterminer quand accorder l'accès du bus est complètement spécifique de la conception. Là où il y a plus d'un maître du bus dans le système, nous devons expliquer la possibilité des conflits sur le bus. C'est à dire, si deux maîtres du bus demandent le bus en même temps, ou si un maître du bus demande le bus tandis qu'il est en service, alors il y aura retard à accéder au bus. Si le processeur ARM doit accomplir un cycle du bus quand une autre transaction du bus a lieu actuellement, alors elle attendra au moins jusqu'à l'accomplissement de ce cycle du bus avant de pouvoir s'exécuter. Considérer le cas où un processeur lance une certaine activité de

DMA d'une fonction particulière. Il y aurait un degré élevé de corrélation entre l'activité du bus du processeur et du contrôleur DMA, et cela peut ne pas être représentant de l'activité globale du système. Alors certains systèmes sont conçus avec des bus multiples, qui sont configurés pour réduire au minimum ces conflits du bus.

Les réponses « Split » et « Retry » : Dans certaines conditions, un périphérique ne peut pas accomplir une transaction. Sous le protocole du bus AMBA, ce dispositif peut générer une réponse « *split* ». Le processeur ARM peut accomplir d'autres transferts et le périphérique peut plus tard fournir une réponse au cycle du bus. Pour modéliser ce type de transaction, il faut un modèle du périphérique qui peut répondre au processeur de cette façon. Mais cet effet n'est pas facilement modélisé dans un environnement de simulateur de jeu d'instruction. Parce que la réponse de ces signaux peut venir dans un cycle et demi, mais les ISS ne travaillent que sur un cycle entier.

Le RTOS (Real Time Operating System) : Un autre aspect très significatif qui affecte la performance du système est le fonctionnement du RTOS. Pour modéliser les effets du RTOS, on doit l'exécuter d'un mode réaliste. Dans cette expérience on n'a pas inclus le RTOS, cependant, le RTOS pourrait avoir été exécuté dans un ISS et l'environnement de co-vérification. Mais l'encodeur DivX a un OS spécifique.

3.4. L'architecture des sous-systèmes de l'encodeur DivX au niveau RTL

Après le développement du logiciel et la validation d'architecture du l'encodeur DivX au niveau de bus fonctionnel, nous avons procédé à la conception de l'architecture de l'application au niveau RTL. Les composants de matériel fonctionnels ont été remplacés par leurs homologues en RTL et les interfaces fonctionnelles de communication ont été raffinées aux signaux physiques du protocole AMBA.

La réalisation des architectures des sous-systèmes de l'encodeur DivX au niveau RTL fait partie entièrement de mes travaux de thèse. C'est pourquoi initialement le développement des architectures de l'application au niveau RTL était effectué entièrement en utilisant le langage SystemC, pour que ça soit conforme avec les travaux effectués précédemment au niveau système. La conception a commencé par le codage de tous les composants en SystemC constituants les architectures de chaque sous-système en respectant les spécifications du protocole et des composants compatibles avec la norme AMBA données par la documentation publique de la société ARM. Au cours de ces travaux, par l'intermédiaire de la société STMicroelectronics, nous était fournis le kit de développement AMBA. Le kit de développement AMBA présente un environnement pour la conception rapide des systèmes sur puce autour du bus AMBA. Il contient plusieurs descriptions HDL des périphériques compatibles avec le bus AMBA, qui peuvent être

directement configurés et implémentés dans un circuit. Cela nous a permis de concevoir et valider aussi les architectures des sous-systèmes de l'encodeur DivX en utilisant le langage VHDL.

Chaque sous-système de l'encodeur DivX était conçu autour d'un processeur ARM946E-S et une architecture organisée autour du bus AMBA. L'architecture du bus AMBA se compose d'un bus à rendement élevé, appelé AHB et un bus périphérique appelés APB. Les bus AHB et les APB sont reliés par l'intermédiaire d'un pont. Plusieurs esclaves sont reliés au bus AHB comme : l'esclave par défaut « DefaultSlave » utilisé pour répondre aux transferts adressés aux régions de la mémoire non définies, où aucun esclave AHB n'est implémenté ; le contrôleur d'interruption « IRQ Ctrl » fournissant l'interface pour générer les deux interruptions du processeur ARM9E-S à partir des sources d'interruption multiples ; et un pont AHB vers APB connecté comme un esclave sur le bus AHB, qui est un maître sur le bus APB. Plusieurs périphériques sont attachés autour d'un bus APB comme : le « Timer » permettant d'accéder à deux décrémenteurs programmables de 32 bits ; la périphérique « WatchDog » servant à surveiller un événement particulier et agissant en cas de dysfonctionnement du logiciel ; et le contrôleur « Remap/Pause » requis pour implémenter le démarrage correcte et pour mettre le processeur dans un mode de basse puissance. Tous les contrôleurs de périphériques font partie d'un ensemble des composants fournis pour la conception des systèmes monopuce autour du bus AMBA. Les caractéristiques principales de ce protocole sont présentées dans les détails dans la spécification AMBA 2.0 [ARM99].

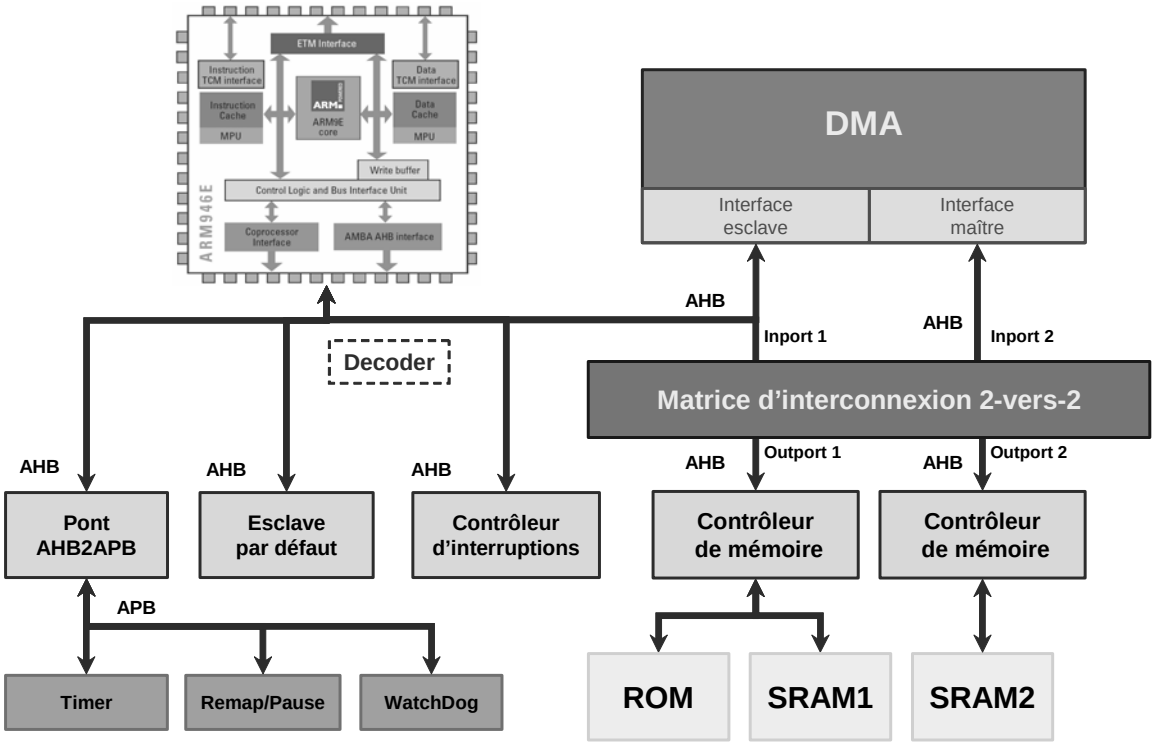


Figure 5.9 : L'architecture du sous-système DivX.

L'architecture du sous-système DivX (voir figure 5.9) possède un système mémoire spécifique contenant deux bancs des mémoires SRAM pour permettre les transferts de données vidéo concurrents à partir du contrôleur DMA et du processeur. Cette fonctionnalité est réalisée par l'insertion d'un composant de matériel spécifique nommé « BusMatrix ». Il est défini dans la documentation du bus AMBA multi couche [ARM02].

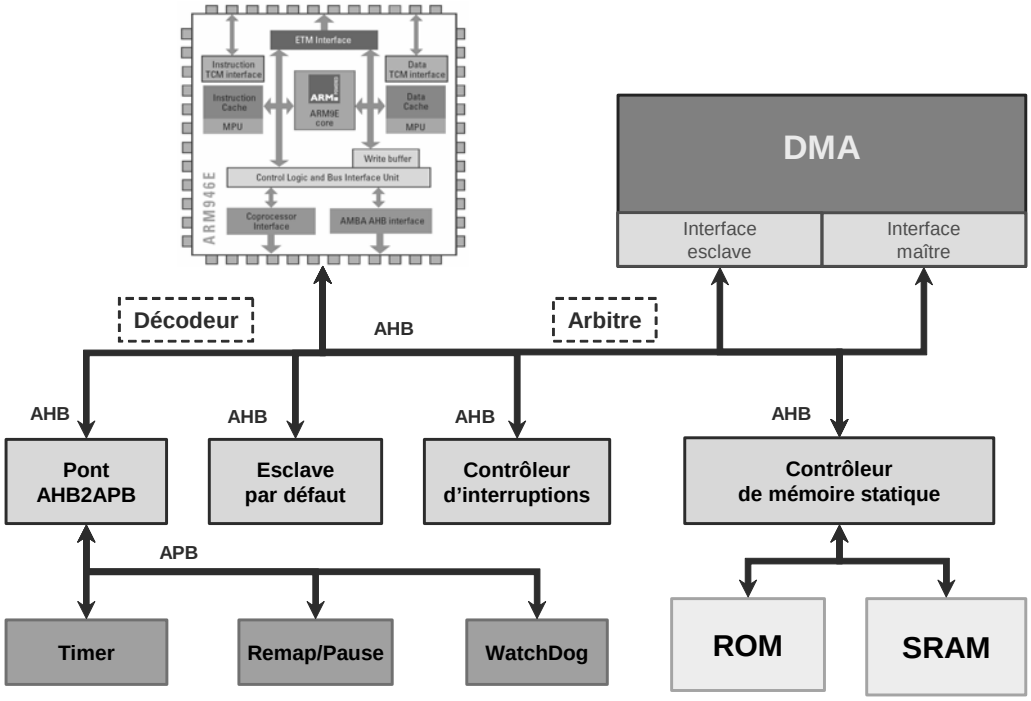


Figure 5.10 : L'architecture du sous-système VLC.

La figure 5.10 présente l'une des deux versions de l'architecture du sous-système VLC. La version présentée est l'architecture finalement implémentée dans l'application DivX. La deuxième version présentée plus loin (cf. page 143) dans le manuscrit est légèrement modifiée pour pouvoir vérifier et faire l'estimation de la puissance consommée de cette architecture. En général, ce sous-système possède la même architecture de base que le sous-système DivX, incluant seulement un banc de mémoire simple pour stocker les données avant le traitement.

3.5. Les particularités de certains composants matériels

Comme présenté dans le paragraphe précédent, chaque sous-système de l'encodeur DivX est conçu autour d'un processeur ARM946E-S et possède plusieurs périphériques interconnectés par un bus AMBA. C'est pourquoi on va donner un peu des détails sur la version du processeur qu'on a choisi d'implémenter dans notre système monopuce multiprocesseur. Nous allons détailler également les différents composants matériels constituant notre architecture. En effet ces particularités doivent apparaître dans la description au niveau RTL, notamment dans le programme d'amorce (intialisation).

3.5.1. Le processeur ARM946E-S

Le processeur ARM946E-S est membre de la famille des processeurs ARM9E-S. L'architecture du processeur ARM946E-S contient le cœur du processeur ARM9E-S, les caches d'instructions et de données, des contrôleurs de commande pour le processeur et des interfaces pour communiquer directement avec un processeur DSP, avec des périphériques externes d'un bus AMBA AHB et des mémoires TCM (voir figure 5.11). L'architecture du processeur ARM946E-S est une architecture Harvard.

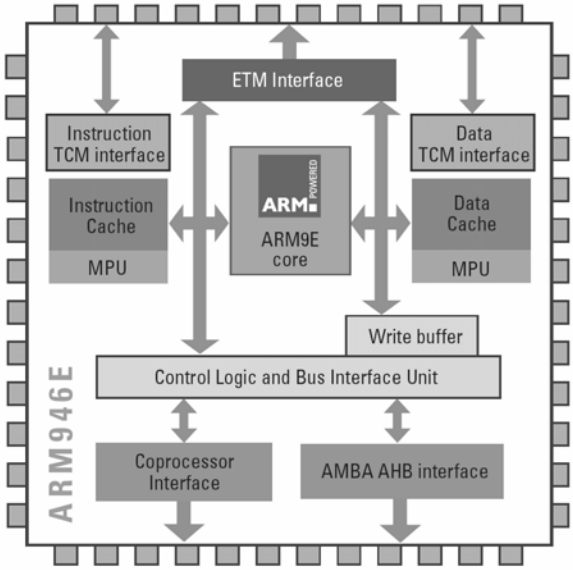


Figure 5.11 : ARM946E-S –L'architecture ARM v5TE avec un cœur ARM9E-S, des caches et des contrôleurs de commande et des interfaces.

Le contrôleur de commande : Le contrôleur du système surveille des interactions entre le cache d'instruction, la mémoire d'instruction, le cache des données, la mémoire des données et l'interface de bus AHB. Il contrôle l'arbitrage interne entre les transferts des blocks de mémoire et des caches.

Le registre CP15 (registre de co-processeur) : Le CP15 permet de configurer les tailles des deux caches et les deux mémoires TCM et aussi de spécifier d'autres fonctions du processeur ARM946E-S.

Quelques registres de CP15 sont programmables par l'utilisateur qui donne la possibilité sur des options comme :

- ✓ Big ou Little Endian
- ✓ Etat de basse puissance (consommation).
- ✓ Partitionnement et protection de la mémoire
- ✓ Test de la mémoire BIST (Built-In Self Test)

L'unité de protection : L'unité de protection permet de partitionner les mémoires et de spécifier des attributs individuels pour chaque région de la mémoire. Les deux espaces d'adressage des instructions et des données peuvent être divisé en 8 régions avec des largeurs variables. Les attributs de l'unité de protection pour chaque région de la mémoire spécifient les propriétés des accès comme : « *cachable* », « *bufferable* », *accès utilisateur ou accès superviseur.*

Les caches : Deux caches sont implémentés dans le processeur ARM946E-S, un pour les instructions et un pour les données, les deux ont une largeur de ligne de huit mots. Chaque cache est construit à partir d'une mémoire SRAM connecté avec le cœur du processeur par un bus de 32 bits qui donne l'accès à une instruction à chaque cycle d'horloge.

Une fonction permettant d'enfermer une partie des instructions dans les caches est fournie pour des applications avec une séquence de code critique.

La mémoire tampon (Write Buffer) : Le processeur ARM946E-S possède aussi une mémoire-tampon d'écriture, pour éviter une interruption du processeur pendant l'accès d'un transfert sur le bus externe.

Les mémoires TCM (Tightly Coupled Memory) : Ce sont des mémoires SRAM intégrés dans l'architecture locale du processeur ARM946E-S, qui ont un accès d'un cycle d'horloge. D'après la spécification des ARM9, il peut gérer jusqu'à 1MB.

La macro du processeur ARM946E-S est préconçu pour la technologie HCMOS9GP de la société STMicroelectronics avec les caractéristiques données dans le tableau suivant :

	Surface (mm^2)	Puissance statique (mW)	Puissance dynamique (mW/MHz)	Fréquence (MHz)	Cache (Bytes)	TCM (Bytes)
CMOS 0.12µm						
ARM946E-S	**2.29**	**0.5**	**0.34**	**216**	**4K / 4K**	**32K / 32K**

Le processeur ARM946E-S est simulé un utilisant une description de simulation sous le format « SwiftModel » et peut être exécuté par plusieurs simulateurs industriels de matériel. Cette description peut être annotée en utilisant le fichier fourni avec les « timings » du processeur sous format SDF, extrait de la netlist après la fabrication du processeur. Cela rend la description du processeur temporellement précis (time accurate) et elle se comporte comme un vrai processeur.

3.6. Le commutateur « BusMatrix »

Le commutateur « BusMatrix » permet l'accès parallèle à un certain nombre d'esclaves AHB partagés d'un certain nombre de différents maîtres AHB. Un commutateur détermine quel esclave particulier exige l'accès à un maître et conduit le port d'entrée à l'esclave approprié. En utilisant le « BusMatrix », il n'y a aucun besoin de mettre en application un arbitre pour permettre l'accès

distribué dans le temps sur le bus AHB. Ainsi dans le cas de l'architecture du sous-système DivX, le processeur ARM946E-S est le seul maître sur le bus AHB et il est tout le temps accordé. Le commutateur « BusMatrix » permet au processeur et au contrôleur DMA de fonctionner séparément en parallèle avec les deux bancs de mémoires. Nous avons nous même conçu le commutateur « BusMatrix » en SystemC ainsi que le reste des composants qui seront présentés ensuite. Les versions initiales du commutateur « BusMatrix » et des autres composants décrits en SystemC étaient comparés avec les versions VHDL qui nous était fournis avec le kit de développement du bus AMBA.

3.6.1. Le temporisateur « Timer »

Le module « Timer » est un esclave APB, permettant d'accéder aux deux compteurs FRC (Free Running Counters) programmables de 32 bits générant des interruptions.

Les sections principales du module de temporisateur sont :

- ✓ deux instanciations identiques d'un compteur courant libre programmable de 16 bits,
- ✓ génération d'interruption basée sur une valeur de compteur.

Le temporisateur possède deux modes de fonctionnement :

- ✓ le mode libre : Le compteur décrémente jusqu'à zéro, réinitialise et continue de décrémenter à partir de la valeur maximale, programmée dans l'un des ses registres. C'est le mode par défaut.
- ✓ le mode périodique : Le compteur produit une interruption à un intervalle constant, rechargeant la valeur originale après passage par zéro.

3.6.2. Le contrôleur d'interruption

Le contrôleur d'interruption est un esclave AHB, fournissant une interface logicielle simple pour gérer les interruptions du système. Il se compose d'un registre statut et un registre pour le statut des sources d'interruptions générées et de deux registres séparés pour « set » et « clear » des registres pour permettre le traitement indépendant de chaque source d'interruption. Le contrôleur d'interruption intégré dans chaque sous-système possède les caractéristiques suivantes : des interruptions sensibles au niveau du signal et des sources d'interruptions programmables.

3.6.3. Le pont APB

Le pont APB est un esclave AHB, fournissant une interface entre le bus à haut rendement AHB et le bus de basse puissance APB. Les transferts de lecture et d'écriture sur le bus AHB sont convertis en transferts équivalents sur le bus APB.

3.6.4. L'arbitre

Le bus AMBA est une norme de bus multi maître. Alors, un arbitre de bus est nécessaire dans le cas de l'architecture du sous-système VLC pour s'assurer que seulement un maître à accès sur le bus à n'importe quel moment particulier. Chaque maître du bus demande la commande du bus et l'arbitre décide ce qui a la priorité la plus élevée et génère un signal « grant » en conséquence.

L'arbitre intégré dans les sous-systèmes de l'application DivX possède les caractéristiques suivantes :

- ✓ capacité de soutien des transferts suspendus, « split transfers »,
- ✓ capacité de soutien des transferts verrouillé, « locked transfers »,
- ✓ Schéma d'arbitrage fixe.

3.6.5. Le décodeur d'adresses

Le décodeur est utilisé pour décoder le bus d'adresses et pour produire des lignes de sélection pour chacun des esclaves du système. Ce module contient seulement de la logique combinatoire pour décoder le bus d'adresses de système, en utilisant le signal « remap » pour commander le choix de la mémoire interne et externe.

Le décodeur commande la carte mémoire du système et produit d'un signal de sélection de l'esclave pour chaque région de mémoire

Le signal de « remap » est utilisé comme un moyen pour fournir une carte mémoire différente au moment de démarrage du système, quand la mémoire ROM se trouve à l'adresse 0x00000000 et au moment d'opération normale quand la mémoire RAM interne devrait être retrouvée à l'adresse 0x00000000.

Le signal « remap » est typiquement fourni par un périphérique Remap/Pause, qui conduit le signal « remap » bas (logique 0) après la mise sous tension du circuit.

3.6.6. Le contrôleur de « Remap/Pause »

Le contrôleur « Remap/Pause » intègre deux fonctionnalités : la fonction de « pause » et la fonction de « remap » de la carte mémoire, qui sont décrites en suite.

La fonction « pause » : La fonction « pause » définit simplement une méthode pour permettre au processeur du système d'entrer dans un état de basse puissance en attente d'une interruption pour le réveiller, où le système n'exige pas du processeur d'être en activité.

Une écriture dans le registre « pause » met le processeur en « attente pour une interruption ». L'effet exact de cette fonctionnalité n'est pas défini dans la documentation des

processeurs ARM, mais typiquement il empêcherait le processeur de chercher des instructions complémentaires jusqu'à ce qu'il reçoive une interruption.

La fonction de « remap » de la carte mémoire : La fonction de « remap » de la carte mémoire fournit une méthode pour remplacer pour un certain période de temps la mémoire principale qui ne contient pas des instructions valides au moment du démarrage du système avec une mémoire contenant des instructions pour initialiser le systèmes et de copier les instructions valides dans la mémoire principales. Une écriture dans un des registres de la fonction « remap » cause le changement de la carte de mémoire du système de celle exigé après le démarrage du système avec celle exigée pendant l'opération normale du système. Une fois que la carte mémoire de reset a été changée et la carte mémoire normale est en service, il ne sera pas possible de reprendre la carte mémoire de reset, autrement que d'effectuer un redémarrage du système.

Une implémentation typique de ce périphérique trouve place dans un système monopuce pour placer la mémoire ROM du système temporairement à l'adresse 0x00000000 au moment du démarrage au lieu d'une mémoire RAM plus rapide qui se trouve constamment à cette adresse, et de changer la carte mémoire une fois que l'opération normale du système peut commencer. Dans un système où une telle fonction de « remap » ne se produit pas l'écriture dans les registres du module n'aura aucun effet.

3.6.7. L'esclave par défaut

L'esclave par défaut est un circuit « dummy », utilisé pour répondre aux transferts qui sont faits aux régions non définies de la mémoire, où aucun esclave de système n'est connecté.

3.6.8. Le contrôleur de la mémoire statique

Le contrôleur de mémoires statiques sert à interconnecter le bus des mémoires intégrés dans le système avec le bus AMBA-AHB. Le contrôleur mémoire implémenté dans nos sous-systèmes permet le raccordement jusqu'à sept bancs de 256MB de mémoire statique, par exemple SRAM et ROM, avec un bande passante de 32 bits. Il est implémenté avec un cycle d'attente supplémentaire pendant un accès en lecture et avec au minimum deux cycle d'attente supplémentaires pendant un accès en écriture. Cela est exigé pour éviter l'appariation des « glitch » dans le fonctionnement du contrôleur.

3.6.9. La mémoire SRAM

Pour notre application nous avons choisi d'intégrer des mémoires statiques qui étaient à notre disposition via l'organisation CMP. Ces mémoires comme le processeur ARM, nous étaient fournis avec des descriptions simulables, que nous avons pu intégrer dans notre environnement de

vérification. Ces mémoires fonctionnent en mode synchrone avec le front montant du signal de l'horloge. Les caractéristiques principales du dispositif sont :

- ✓ une opération entièrement synchrone
- ✓ une optimisation pour la vitesse, la surface et la puissance, par ordre d'importance
- ✓ une cellule de mémoire de six transistors, en opération entièrement statique.

Nous avons implémenté pour chacune des deux bancs de mémoire du sous-système DivX, deux blocs mémoires de 16 kB (4096x32 mots) de la mémoire SRAM, type SPHS9GP de la société STMicroelectronics [SPH02].

3.6.10. Le module « Splitter »

Le composant « Splitter » sert à découper l'image entrant du flux de vidéo, en tel nombre des parties comme le nombre des sous-systèmes DivX traitant l'image. L'objectif principal de ce composant est de déterminer la position et l'orientation de chaque pixel de l'image entrant en appliquant un algorithme de détection par rapport à la taille d'image. La figure 5.12 donne le schéma bloc du composant Splitter.

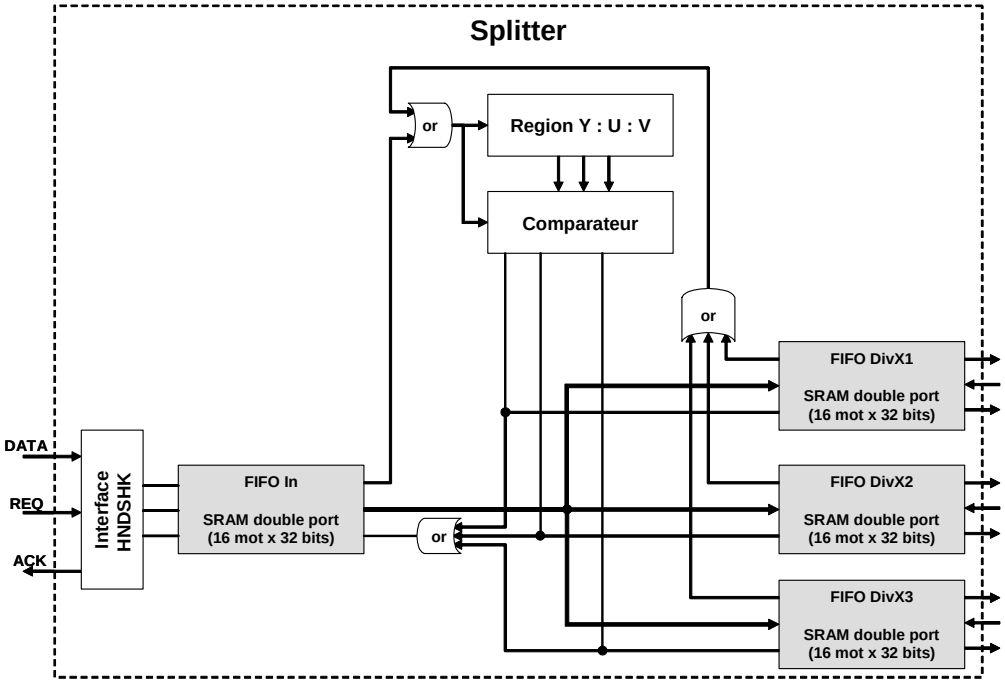


Figure 5.12 : Le schéma bloc du composant Splitter

Chaque pixel de l'image entrant est stocké temporairement dans une FIFO à l'entrée, puis par rapport à sa place il est envoyé vers une des FIFO destinées pour chaque sous-système DivX. L'implémentation de ce module est réalisée pour nos besoins, elle est configurable et peut être élargie. L'inconvénient principal est le nombre des FIFO nécessaires à insérer par rapport au nombre des sous-systèmes utilisés.

3.6.11. Le module « Combiner »

L'image compressée par le sous-système VLC est transférée vers un module nommé « Combiner ». Le rôle du composant « Combiner » est de réordonner les données reçues du sous-système VLC, en récupérant la valeur (ou la longueur) de la séquence des données qui doivent être envoyer vers un module de stockage. Une deuxième fonctionnalité du « Combiner » consiste à retrouver la valeur quantitative ou « quanta » et de l'envoyer vers le module « Splitter » comme montre sur la figure 5.13 :

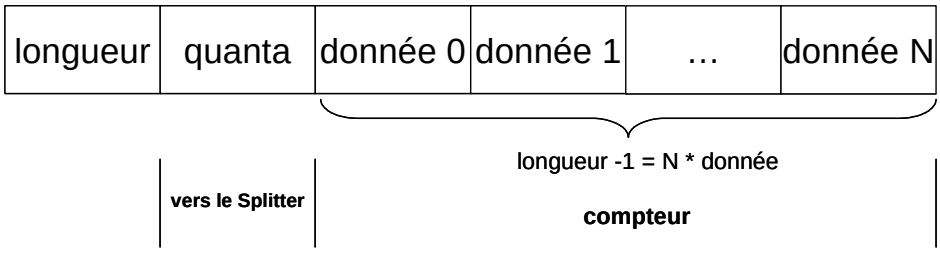


Figure 5.13 : Le principe de fonctionnement du composant « Combiner ».

L'implémentation du composant « Combiner » est présentée sur la figure 5.14.

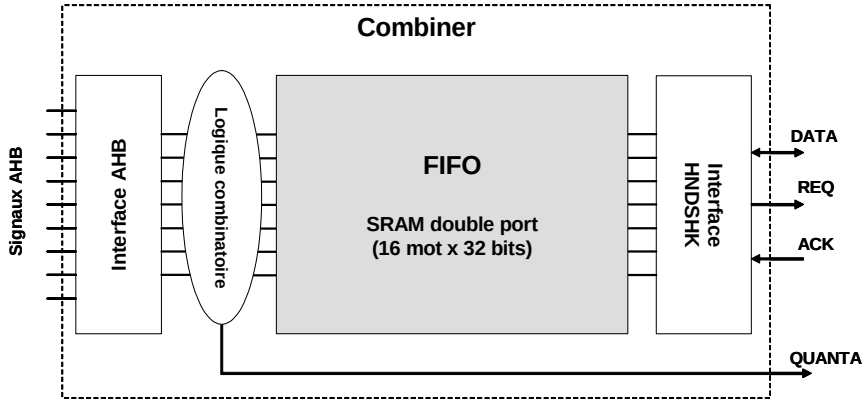


Figure 5.14 : Le schéma bloc du composant « Combiner ».

Elle contient une interface esclave AHB du protocole AMBA pour permettre le transfert direct à partir du processeur du sous-système VLC et une interface « handshake » pour être interconnecté avec une module externe de stockage. L'architecture du « Combiner » contient aussi une logique combinatoire réalisant la fonctionnalité décrite ci-dessus et une FIFO réalisée avec une mémoire SRAM à double port de 16 mot par 32 bits.

3.6.12. Le contrôleur DMA

Le module met en application un contrôleur DMA programmable, que peut effectuer des transferts de type mémoire à périphériques et des transferts de données de mémoire à mémoire. Dans le contrôleur DMA sont intégrés quatre canaux programmables indépendamment l'un de l'autre. Le contrôleur supporte aussi des requêtes de transfert par l'intermédiaire du matériel ou du

logiciel. La réalisation matérielle de ce contrôleur de DMA fait partie de mes travaux de thèse. Le contrôleur de DMA est implémenté en suivant les spécifications de deux contrôleurs : le contrôleur de DMA 8237A [INT93] de la société Intel et le contrôleur de DMA PL080 [DDI04] de la société ARM. Ce contrôleur était codé au niveau RTL en SystemC et en VHDL. Il était aussi intégré dans la bibliothèque de l'outil MaxSim, décrit au niveau de bus fonctionnel avec SystemC.

Les type de transfert : Comme déjà noté le contrôleur de DMA effectue deux types de transfert :

- ✓ Mémoire à mémoire : le DMA lit un mot de la mémoire en utilisant l'interface de son port maître « ahb1master » (voir la figure 5.15) et écrit le même mot par l'interface de son deuxième port maître « ahb2master »,
- ✓ Mémoire au périphérique : le DMA lit un mot avec par l'une des interfaces maître, le port ahb1master ou ahb2master, et écrit le mot en utilisant le même port.

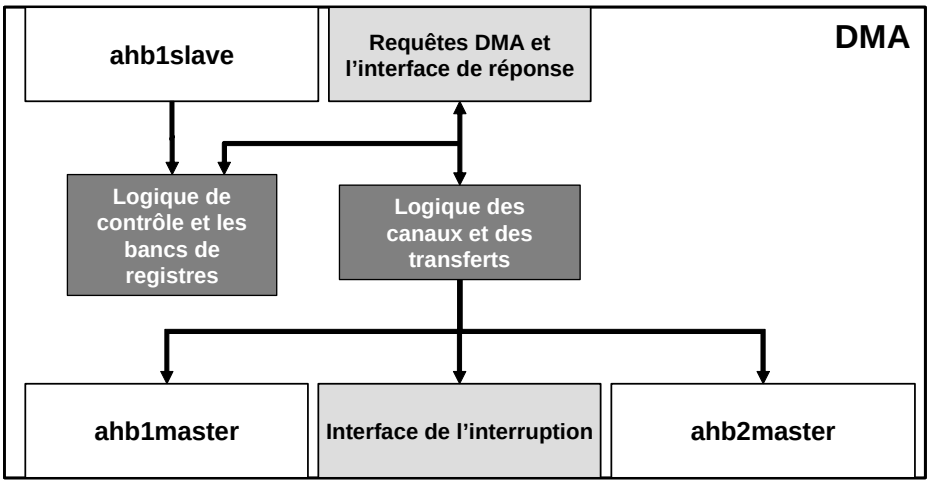


Figure 5.15 : Le schéma bloc du contrôleur de DMA.

Le contrôleur d'accès direct à la mémoire contient deux interfaces maîtres conçues d'après le protocole AHB de la norme AMBA. Elles servent pour effectuer les transferts de données à partir d'une mémoire vers une mémoire, et à partir d'une mémoire (ou périphérique) vers une périphérique (ou mémoire). L'interface du port esclave est utilisée seulement pour configurer le contrôleur de DMA et pour générer des requêtes de transferts par logiciel. Le contrôleur DMA possède également une interface pour recevoir et répondre aux requêtes de transferts générés par des périphériques externes. Il possède également une interface des interruptions pour synchroniser et signaler le début ou la fin d'un transfert à un processeur

La programmation : Il faut programmer plusieurs registres de DMA avant de commencer à transférer des données. Cependant, pour éviter un comportement imprévisible, il est obligatoire de désactiver tous les transferts courant du contrôleur DMA avant de le programmer en écrivant dans son registre de commande.

Les registres :

- ✓ Registres d'adresse de base : chaque canal a deux registres de base, qui contient l'adresse initiale de la source et l'adresse initiale de destination pour les transferts de DMA,
- ✓ Registres d'adresse courante : chaque canal a deux registres d'adresse courant (pour la source et pour le destinataire) qui contient la valeur d'adresse du transfert courant de DMA,
- ✓ Registres de mots de base : le même objectif comme les registres d'adresse de base, mais ils contiennent le nombre des mots à transférer,
- ✓ Registres de mot courant : les mêmes que les registres d'adresse courant,
- ✓ Registre de commande : le registre de commande configure l'opération de DMA, tel que la schéma de priorité, l'activation ou désactivation de fonctionnalité du DMA etc.,
- ✓ Registre de mode : contient la configuration pour chacun des quatre canaux du contrôleur d'accès direct à la mémoire,
- ✓ Registre de requête : pour permettre des requêtes de transferts par logiciel,
- ✓ Registre de masquage : ce registre peut être configuré pour désactiver une requête de transfert entrante de DMA,
- ✓ Registre statut : contient les drapeaux de statut de DMA.

Les modes de transfert : Le contrôleur de DMA possède trois modes de transfert :

- ✓ le mode simple : un transfert de DMA seulement est exécuté et la machine d'état fini est réinitialisée,
- ✓ le mode bloc : les transferts de DMA continuent sans interruption jusqu'à ce que le transfert soit accompli ou qu'une requête de transferts avec la priorité plus élevée arrive,
- ✓ le mode de demande : les transferts de DMA peuvent être interrompus en désactivant le signal d'entrée de requête d'un périphérique.

4. Le prototypage physique de l'encodeur DivX

La vérification des systèmes monopuce devient de plus en plus difficile. Nous devons vérifier non seulement le fonctionnement correct de tous les composants matériels mais également le protocole de communication et l'intégration du logiciel sur la puce. Pour améliorer la qualité de la vérification pour répondre à nos exigences, nous devons employer des plateformes de vérification avancées dans nos projets à un coût raisonnable.

4.1. La plateforme de vérification

La plateforme de vérification est une méthode utilisée pour faire fonctionner une description de matériel. Le processus de conception décrit le matériel en utilisant des langages de description matérielle tels que Verilog ou VHDL, ou un langage de plus haut à niveau, tel que SystemC. Cette représentation de HDL d'un système peut être simulée en utilisant diverses plateformes d'exécution ou environnements de prototypage.

Il y a trois méthodes de base pour la validation utilisée dans le flot de conception physique d'un système monopuce décrit en détails dans [AND04]. Nous les reprendrons brièvement :

Le prototype matériel se rapporte à la construction d'un prototype du système cible en utilisant des composants discrets comme FPGA et mémoires. Le prototype est une représentation du système final qui peut être construit plus rapidement et il est disponible plus tôt que le produit réel. Pour gagner du temps pendant le prototypage, l'architecture matérielle du système est basée sur des composants de matériel réutilisables où la partie ASIC est remplacée par la logique programmable.

L'émulation matérielle utilise le processus de la synthèse et la mise en place automatique de la description matérielle entière d'un système monopuce dans une plateforme de matériel qui effectue une exécution en parallèle pour augmenter la vitesse de la simulation. Il existe deux types différents d'émulation : l'émulation par activation et observation de la plateforme matérielle par un poste de travail, et l'émulation « en circuit » par activation et l'observation directes par l'intermédiaire du matériel externe couplé à la plateforme.

Le but de cette méthode de vérification est de fournir un environnement plus réaliste et plus rapide pour la conception. Un exemple pour l'émulation en circuit est la plateforme ARM Integrator [SAS03] pour le prototypage rapide des systèmes monopuce.

La simulation de matériel est le processus de vérification de matériel sans synthétiser la description HDL du système. La simulation logique est basée sur un simulateur événementiel qui fonctionne par la propagation des changements des entrées par le circuit jusqu'à ce qu'un état d'équilibre soit atteint. Le simulateur logique fonctionne sur un poste de travail en utilisant le langage de description de matériel pour décrire la conception et les vecteurs de test. Le logiciel du projet est implémenté comme un ensemble de bits dans une mémoire. Cette mémoire a des caractéristiques temporelles traité par le simulateur.

Nous avons choisi pour notre environnement de vérification d'utiliser la technique de la simulation logique. La simulation logique est la plateforme la plus lente de vérification mais elle donne une mesure d'exécution très précise du système – parfois aussi précis que le système réel. La simulation logique est une technique qui combine un modèle exécutable du logiciel et du matériel

du système monopuce. Ce modèle exécutable du système monopuce est habituellement décrit en utilisant un langage HDL. Cette description tient compte de toute l'implémentation et de tous les attributs fonctionnels caractéristiques du système. L'avantage principal d'avoir une telle description du circuit est que le système peut facilement être changé ou optimisé. Changer les nombres d'états d'attente en mémoire, ajoutant un contrôleur périphérique est tout à fait possible dans cet environnement. Un inconvénient de ce modèle est qu'une description entière du système évalué doit être créée dans le langage HDL. Ceci peut être résolu en utilisant des composants IP réutilisables. Cependant, l'inconvénient principal de cet environnement est son exécution : une fois engagé, le temps d'exécution de la simulation est très lent.

4.2. L'environnement de vérification

Une des métriques principales de la qualité de conception de matériel est sa capacité d'exécuter le logiciel du système monopuce. Si la partie matérielle arrive à exécuter toute la partie logicielle du système (comme par exemple : l'amorce, le système d'exploitation et l'application) le circuit doit être libre des bogues (bug).

Ainsi la meilleure vérification doit fournir une combinaison d'un programme logiciel (e.g. le logiciel courant de l'application) et des vecteurs de test utilisés traditionnellement pour vérifier complètement le système sous développement.

Cette section présente deux environnements pour la simulation de matériel et montre comment un environnement de covérification de matériel et de logiciel combinée avec un environnement de génération de stimulus apporte une valeur ajoutée à la vérification au niveau RTL. Nous avons essayé ces deux environnements.

4.2.1. L'environnement de covérification du sous-système VLC.

En utilisant l'architecture de sous-ensemble VLC de l'encodeur DivX et la méthode de covérification, nous avons établi un environnement capable d'exécuter le logiciel sur une description simulable du processeur ARM946E-S et de capter les traces des transferts générés sur le bus AMBA par le programme. La figure 5.16 montre l'environnement de covérification de matériel et de logiciel.

L'environnement de covérification est composé de deux parties. Une partie contentant tous les composants de matériels du sous-système décrits en VHDL au niveau RTL, y compris le processeur ARM946E-S, qui sont simulés avec le simulateur de la société Cadence NC-SIM [CAD04] et une partie générant le programme du logiciel ou plus précisément le fichier « rom.dat » basé sur l'outil ARM ADS « ARM Developpement Studio, version 1.2 » [ARM01]. Le fichier « rom.dat », qui est un fichier binaire dérivé du fichier exécutable du logiciel de l'application, est lu

par une description comportementale d'une mémoire ROM avec des relations temporelles qui remplace la vraie mémoire ROM dans notre environnement de vérification.

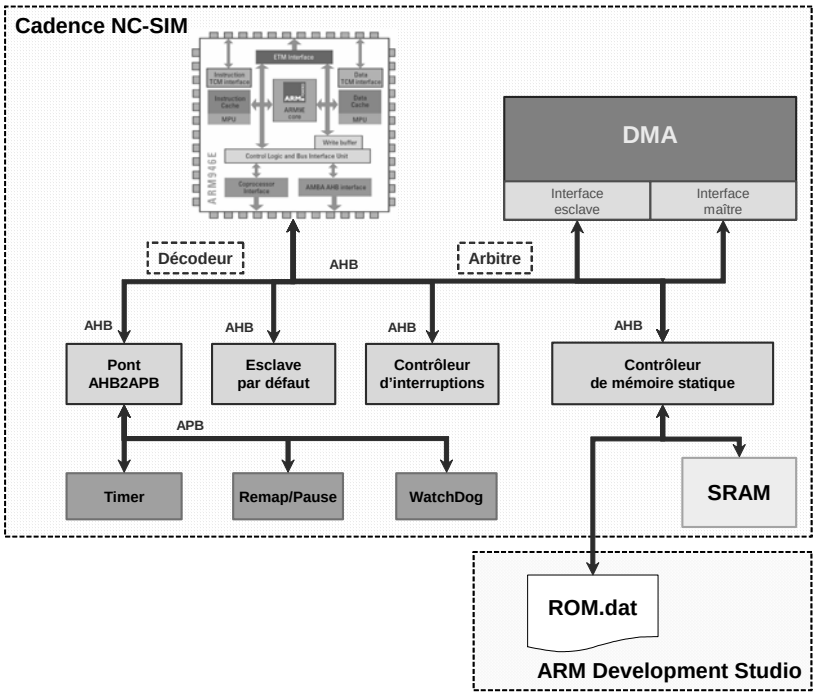


Figure 5.16 : L'environnement de covérification du sous-système VLC.

Cet environnement de covérification est pratique seulement pour de petits programmes. L'exécution résultante de la simulation rend cette plateforme impraticable pour de grands programmes de logiciel, où les concepteurs de logiciel sont principalement intéressés par la vitesse d'exécution pour le processeur et ses instructions.

Nous avons déjà discuté que la plupart des composants constituants chaque sous-système étaient conçus en SystemC par moi-même et en utilisant le kit de développement venant d'ARM. Le présent environnement de vérification concerne la version implémentée par la réutilisation du kit ARM. L'implémentation de l'architecture du sous-système VLC en SystemC est simulée par la technique de simulation propre de SystemC. C'est-à-dire que nous avons utilisé de la même façon l'outil ARM ADS pour générer notre fichier « rom.dat » pour la partie logicielle. Pour la partie matérielle nous avons procédé en utilisant une simulation mixte SystemC et Verilog (pour le processeur) avec le simulateur ModelSim de Mentor Graphics.

4.2.2. L'environnement de vérification avec le FRBM

Un des points difficiles principaux dans la vérification de matériel est le travail manuel exigé pour la création des vecteurs de test. Dans l'environnement de vérification présenté dans le paragraphe précédent, le programme de l'application remplace les vecteurs de test utilisés normalement dans la vérification. Ce qui est l'avantage principal de cet environnement.

L'inconvénient de cet environnement vient quand le test du logiciel retrouve des erreurs qui proviennent d'un problème éventuel dans la description de matériel du système. Alors il est nécessaire que le concepteur du système puisse facilement résimuler le système sans utilisation du programme logiciel mis au point pour l'environnement de covérification de matériel et de logiciel. Dans ce cas le processeur ARM devrait être remplacé par un générateur des transferts qui va faire que des accès simples d'écriture et de lecture sur le bus, comme par exemple le FRBM (File Reader Bus Master).

Des générateurs des transferts comme FRBM sont utilisés depuis longtemps dans la vérification fonctionnelle pour soustraire l'opération des processeurs et pour d'autres interfaces de communication. Dans notre deuxième environnement de vérification utilisé que pour valider la présence (les interconnexions) des composants dans le circuit, nous avons utilisé un programme nommé FRBM fourni avec le kit de conception du bus AMBA.

La figure 5.17 montre l'environnement de vérification avec le FRBM qui a remplacé la description de simulation du processeur ARM. Les transactions dans ce programme sont décrites comme des commandes dans un fichier texte utilisé pour piloter le FRBM qui génère des transactions explicites sur le bus AMBA.

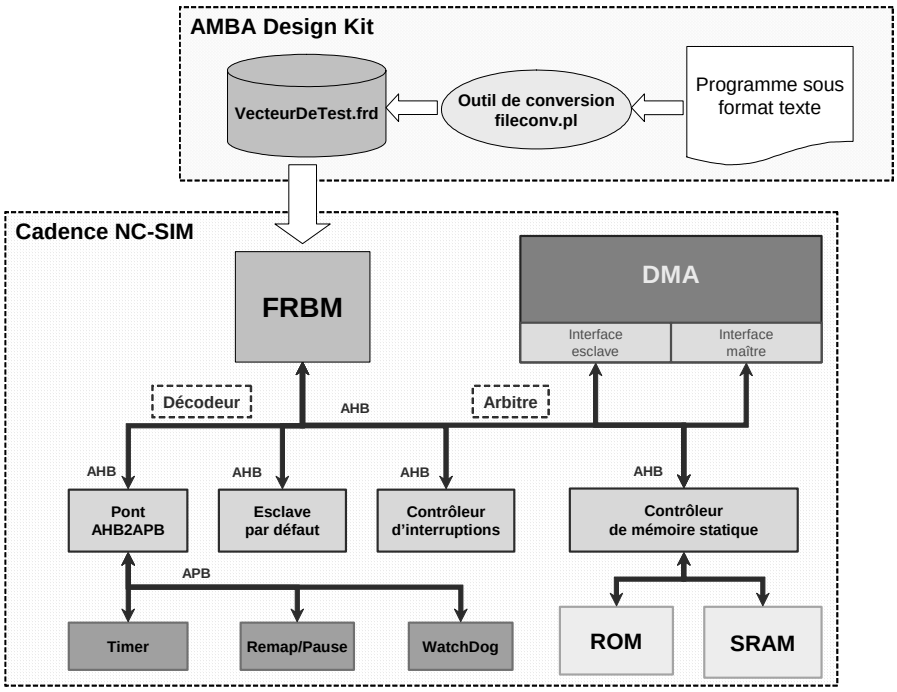


Figure 5.17 : L'environnement de vérification avec le FRBM du sous-système VLC.

Le FRBM permet de générer rapidement des transferts explicites sur le bus AMBA. Il peut fonctionner avec ou sans un processeur présent dans le circuit sous vérification. Un exemple d'un fichier de pilotage contenant des commandes de transaction pour le bus AMBA est montré sur la figure 5.18.

```
#  Nom de fichier  : TestIntegration.fri
#  Objectif        :  Test d'intégration des composants périphériques du sous-système VLC
################################################################################
# Test de la mémoire externe
################################################################################
# Ecrire quatre mots dans la mémoire externe en mode « burst »
 W 31000000  AAAAAAAA   word   incr4
 S           BBBBBBBB
 S           CCCCCCCC
 S           DDDDDDDD
# Insérer deux cycles d'attente
  I
  L 2
# Lire les quatre mots écrits dans la mémoire
  R 31000000 AAAAAAAA   FFFFFFFF   word   incr4
  S          BBBBBBBB
  S          CCCCCCCC
  S          DDDDDDDD
```

Figure 5.18 : Exemple de fichier de commande de FRBM.

Les commandes de transfert dans ce fichier représentent l'action de l'écriture de quatre mot de 32 bits à l'adresse 0x31000000 et la lecture les données. Le FRBM a toute l'information requise pour reproduire l'ordre des opérations comme vu par la conception de matériel sans exiger une description complète du processeur.

4.3. La génération du modèle de simulation

Pour la génération de l'image exécutable du logiciel, nous avons utilisé un flot de conception classique pour un programme décrit en C/C++. Nous avons d'abord compilé et puis assemblé les fichiers objets dans un programme en langage machine.

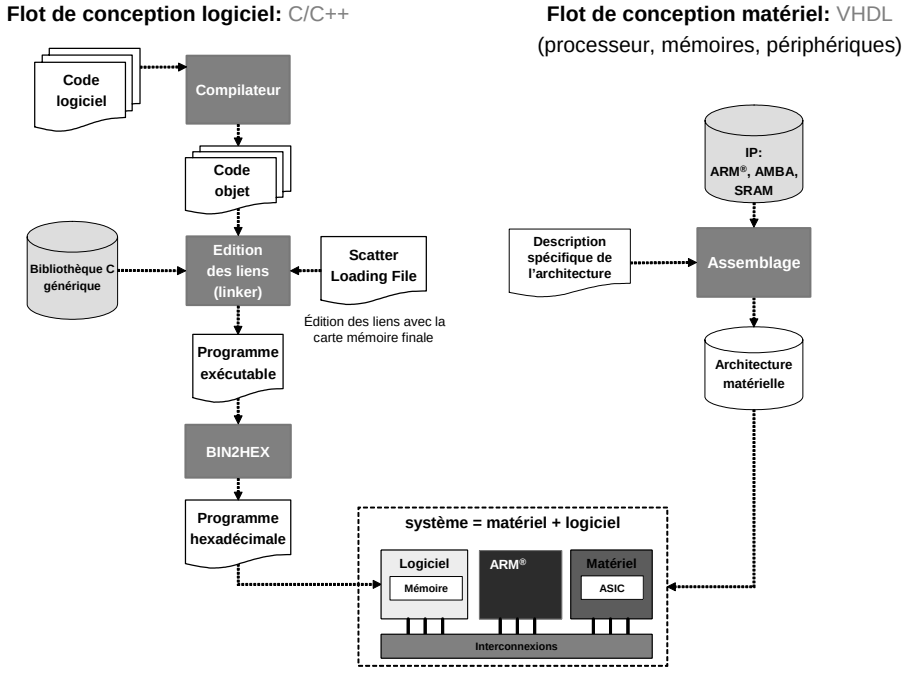


Figure 5.19 : Le flot pour la génération du modèle de simulation.

Une particularité très importante à cette phase est celle que chaque partie code ou donnée du programme assembleur ont été localisées respectant l'exécution de la carte mémoire de l'architecture de matériel décrite à l'aide du fichier « scatter loading ». A ce niveau d'abstraction nous avons appliqué encore une étape : la traduction du programme exécutable en une image de code binaire commode pour l'intégration dans la mémoire ROM. Le flot de vérification pour la génération du modèle de simulation au niveau RTL est montré sur la figure 5.19.

La génération du modèle de simulation pour la partie matérielle du circuit se compose de l'étape d'assemblage des composants dans une description de l'architecture matérielle du système. Ensuite nous avons utilisé le deuxième environnement de vérification pour valider l'intégration de tous les composants avant de procéder à l'étape de validation par l'environnement de covérification de logiciel et de matériel aussi décrit précédemment.

4.3.1. L'implémentation du matériel

Cet environnement de covérification de matériel et de logiciel est basé sur modèle de simulation ou DSM (Design Simulation Model) du processeur ARM946E-S. Le DSM est une description précise du processeur qui peut être incluse directement dans beaucoup de simulateurs de matériel. Dans notre cas nous avons utilisé le simulateur Cadence NC-Sim [CAD04].

Le reste de l'architecture matérielle du sous-système VLC de l'encodeur de DivX au niveau RTL a été réalisé par la réutilisation des composants IP existants tels que les périphériques disponible dans le kit de conception AMBA fournissant un environnement générique pour la conception rapide et une bibliothèque avec des mémoires SRAM synchrones.

4.3.2. L'implémentation du logiciel

La difficulté principale de l'implémentation du logiciel dans la conception d'un système monopuce est la carte mémoire, connue après l'implémentation finale du matériel. Pour simplifier le flot de conception des systèmes monopuce, nous avons proposé à cette phase d'intégration d'utiliser les mécanismes qui permettent d'indiquer la carte mémoire finale pendant le processus d'édition des liens du logiciel. Ainsi nous serons capables de décrire chaque région dans le code d'image qui a une adresse d'initialisation et d'exécution différentes dans le système de mémoire.

Un des mécanismes permettant ceci est le fichier de description de chargement et de placement de la mémoire [ARM98] qui était précédemment discuté dans le chapitre 3. La technique de chargement et de placement donne le contrôle complet pour grouper et placer des composants d'image dans la mémoire. Nous avons appliqué cette technique pendant la génération du modèle de simulation de notre application au niveau RTL. Cette technique nous a permis de fusionner

facilement le matériel avec le logiciel dans un modèle d'exécution. La figure 5.20 donne l'implémentation de la carte mémoire de notre architecture en utilisant cette méthode.

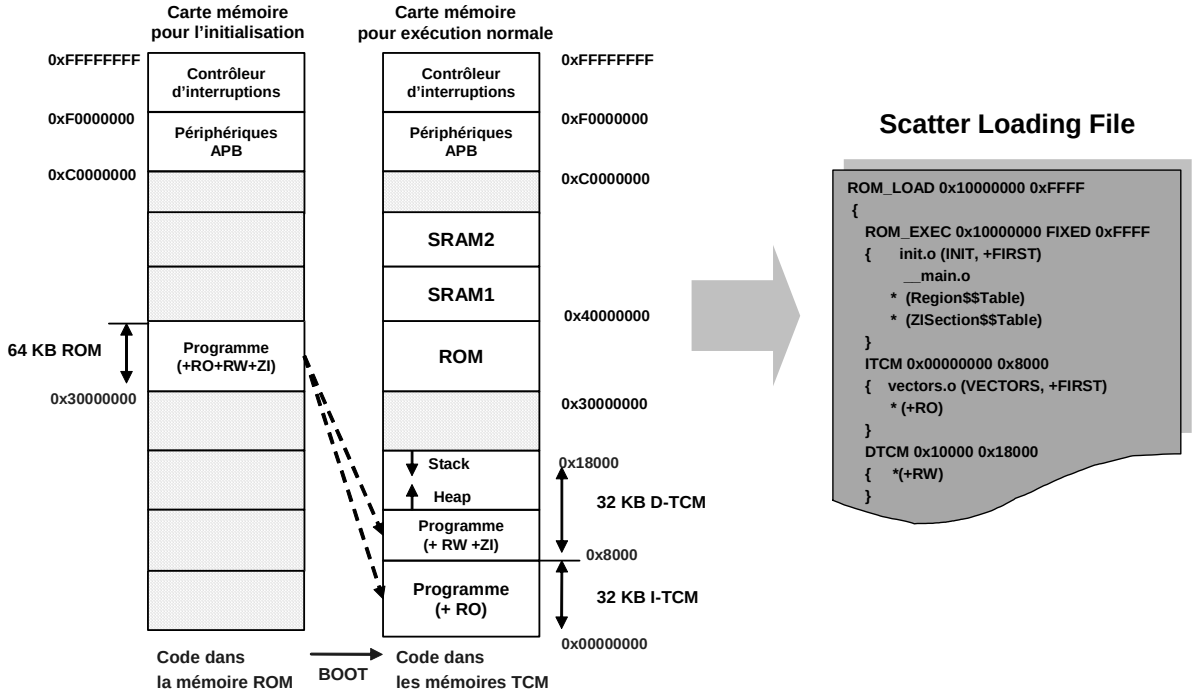


Figure 5.20 : L'implémentation de carte mémoire.

Une mémoire TCM est située à l'adresse 0x00000000 après la mise sous tension, mais il n'y a pas une instruction valide à cette adresse au moment de démarrage. Par conséquent pendant l'initialisation du système, nous devons déplacer les instructions valides vers cette mémoire TCM à partir de la mémoire ROM qui après la mise sous tensions se trouve à l'adresse 0x30000000.

Pour réaliser ce copiage des données, le fichier « scatter loading » décrit, sous un format de texte, le placement de chaque région dans la mémoire au moment après le reset et pendant l'opération normale. Il attribue à chaque région une adresse d'initialisation. Le déplacement du logiciel de la région de chargement à une région d'exécution est fait par une fonction d'initialisation de la bibliothèque C qui fait partie du code amorce du système. Nous travaillons actuellement sur l'adaptation de cette technique dans le processus de conception automatisée du flot ROSES.

Une autre difficulté principale dans la conception du logiciel pour une application embarquée de SoC est l'ordre d'initialisation après le reset du système. Le logiciel d'application doit fournir une initialisation elle-même. Nous voulons préciser qu'au plus bas niveau d'abstraction des systèmes monopuce comme le niveau RTL le code d'initialisation est spécifique pour l'application et le processeur ciblé. Habituellement le code d'amorce effectue l'initialisation et la configuration du processeur, le « remapping » et l'initialisation de la mémoire exigée par code en C, ainsi que l'activation des mémoires caches et les interruptions.

Alors il était nécessaire d'écrire tout le code de l'amorce pour le processeur ARM946E-S, car il n'existait pas et jusqu'à ce moment de conception il était caché par le débuguer du simulateur du processeur.

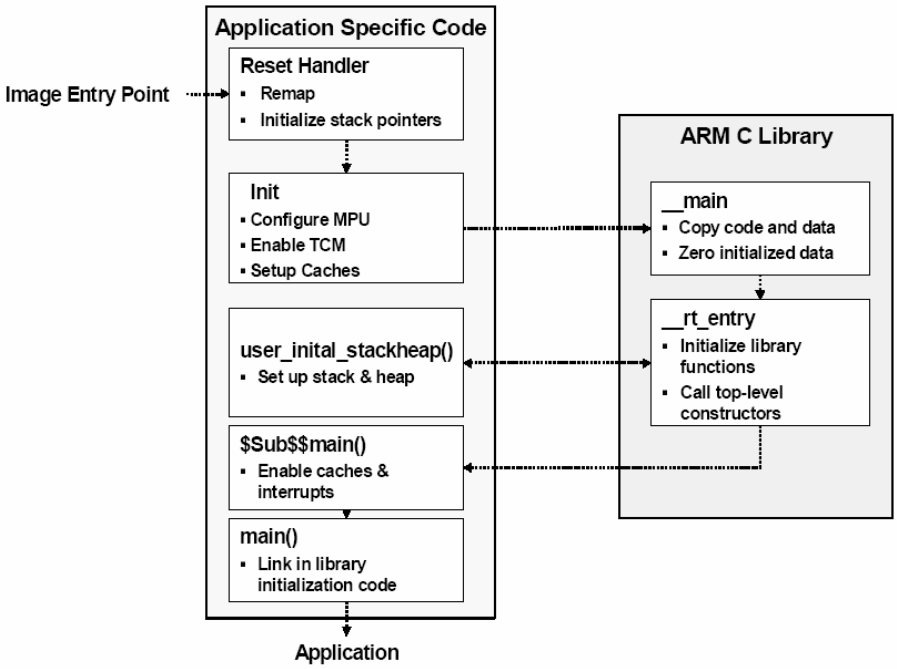


Figure 5. 21 : Le code d'amorce du processeur ARM946E-S.

La figure 5.21 montre un schéma fonctionnel du code d'amorce écrit par moi pour le processeur ARM946E-S et donne une vue d'ensemble des deux parties dans l'ordre d'initialisation de logiciel d'application. Il est développé manuellement par le concepteur de système car c'est une tâche difficile pour l'automatisation. La deuxième partie se compose de la fonction prédéfinie d'initialisation fournie par la bibliothèque C du processeur ciblé. Des parties de ce logiciel peuvent être réutilisées dans des différentes applications.

4.4. Les résultats de simulation

En utilisant le flot de génération du modèle de simulation et les environnements de covérification précédemment présentés, nous avons pu valider l'architecture de chacun des sous-systèmes. Nous avons validé le fonctionnement de chaque composant matériel en combinant le code de l'amorce de l'application DivX avec une séquence des tests pour chaque périphérique. La co-simulation était effectuée sur un poste de travail Sun Blade 150 en utilisant le simulateur « Simvision » de Cadence pour faire fonctionner le programme de logiciel avec l'architecture matérielle du système. Pour la génération du programme de logiciel nous avons utilisé les outils de ARM ADS (ARM Development Studio). La figure 5.2 montre une capture d'écran de la simulation

du code d'amorce du logiciel d'application DivX sur le processeur ARM946E-S. La fréquence d'horloge était fixée à 60 MHz.

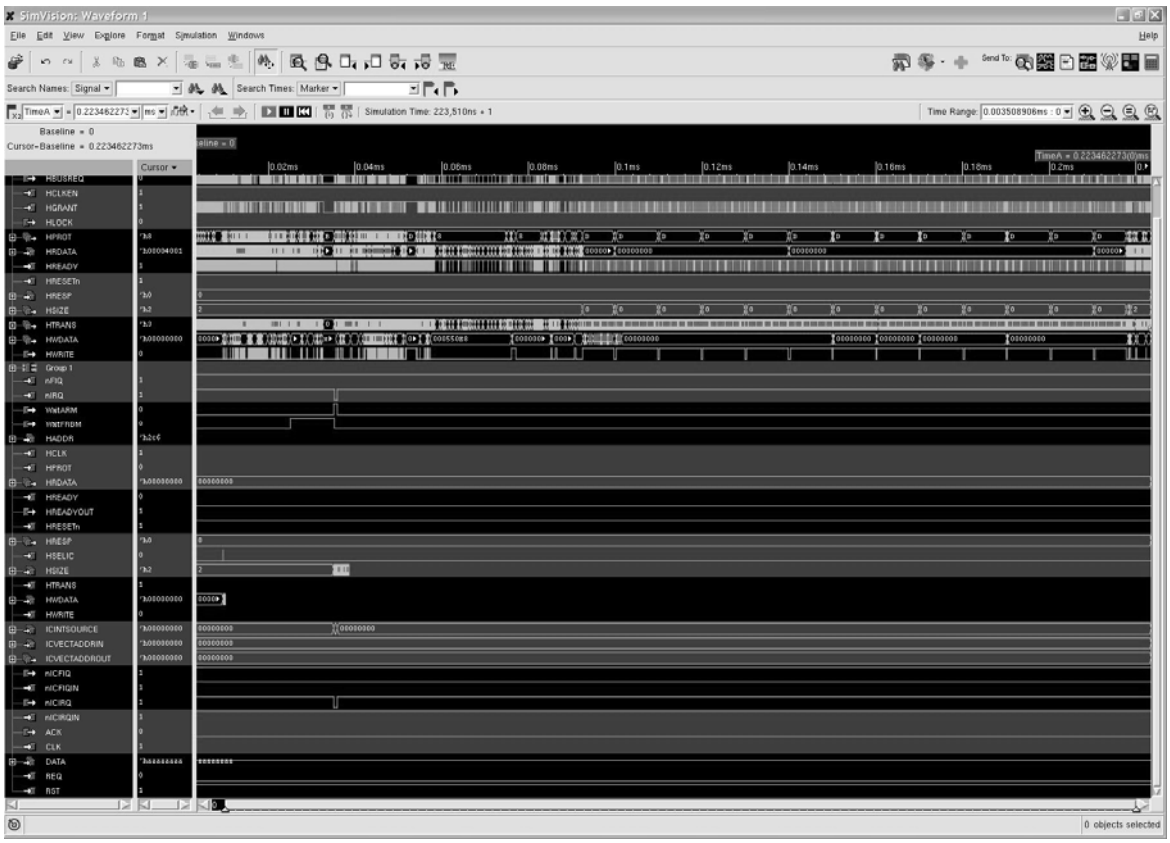


Figure 5.22 : Capture d'écran de la simulation du code d'amorce du logiciel d'application DivX.

Nous avons exécuté tout le programme logiciel nécessaire pour le traitement de la première image d'une séquence de vidéo. Le simulation entière a pris environ une dizaine d'heures pour exécuter environ 1 500 000 instructions. La vitesse approximative de la simulation peut être retenue par le tableau suivant :

```
Code de l'amorce
==================================================
   Total RO    (Code + RO Data)                4320  ( 4.22kB)
   Total RW    (RW Data + ZI Data)              132  ( 0.13kB)
   Total ROM (Code + RO Data + RW Data)        4320  ( 4.22kB)
==================================================

Configuration du processeur ARM946E-S
==================================================
Fréquence d'horloge: 60 MHz

Cache d'instruction : 4096 bytes
Cache données :       4096 bytes

Mémoire TCM d'instructions  : 32 768 bytes
Mémoire TCM données :         32 768 bytes
==================================================

Temps de simulation
==================================================
Temps d'exécution: ~ 83 us
Utilisation de la mémoire : ~ 46.2 M
Utilisation du processeur : ~ 229.2 s
```

Le code d'amorce du processeur est de l'ordre de 4.22 kilo octets (ou environ 7000 instructions), le temps approximatif nécessaire pour le simuler était de 4 minutes.

Nous avons effectué une validation en utilisant l'environnement MaxSim. La figure suivante présente une capture d'écran de cette expérimentation pour l'initialisation du sous-système DivX.

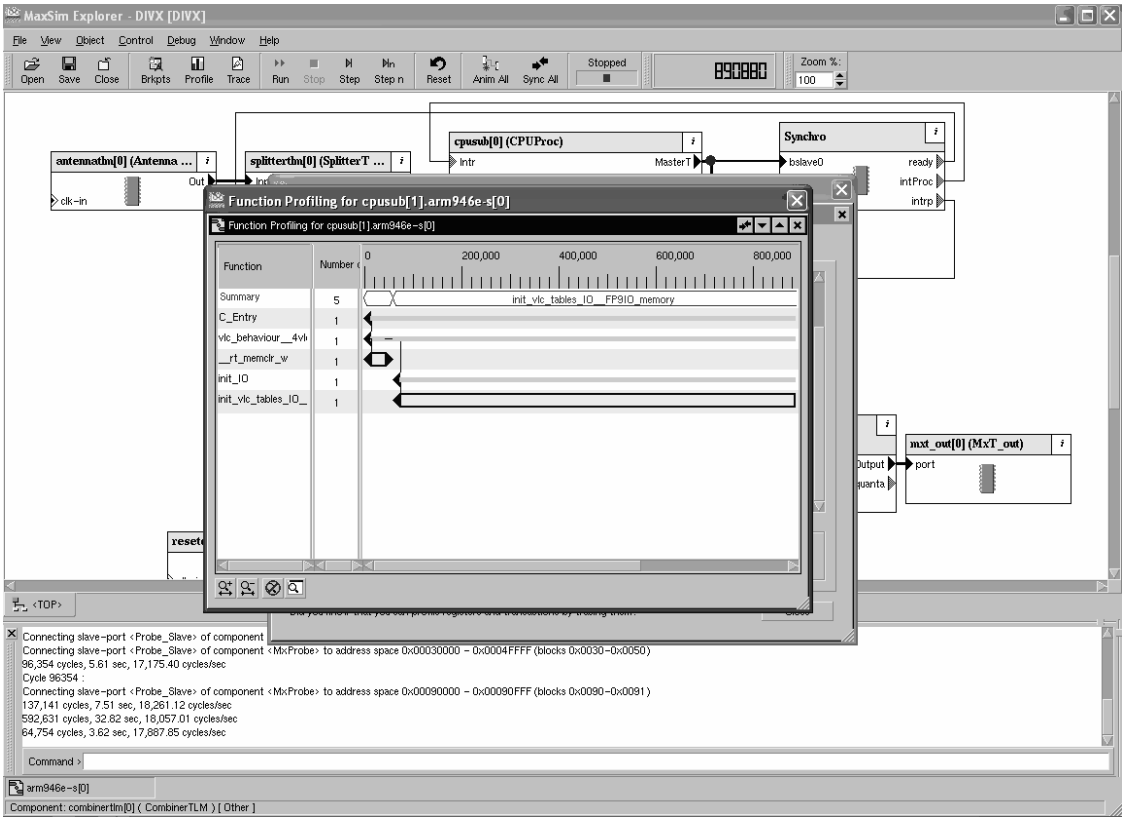


Figure 5. 23 Capture d'écran de la simulation de l'encodeur DivX au niveau de bus fonctionnel dans l'environnement MaxSim.

Pour le même « presque » modèle de simulation que au niveau RTL, c'est-à-dire pour une architecture de matériel de l'encodeur DivX modélisée au niveau de bus fonctionnel en SystemC avec des composants précis au niveau cycle et pour le programme logiciel de l'application utilisé au niveau RTL, nous avons pu exécuter cette représentation en deux heures de temps de simulation.

5. L'étape de la synthèse logique

L'étape de la synthèse logique est le processus capable de générer une netlist de portes logiques, à partir d'une description donnée par les langages HDL. Ce sont les langages qui supportent des concepts spécifiques tels que les concepts de temps, de parallélisme, de communication interprocessus, la réactivité et les types de données spécifiques. Un outil pour la synthèse logique n'accepte en général qu'un ensemble des langages matériels. Les deux langages

largement supportés pour le moment par les outils de synthèse sont VHDL et Verilog. Il y a des travaux pour faire la synthèse logique à partir du langage SystemC, mais pour le moment ces efforts n'ont pas montré des résultats complets.

Pour transformer la description RTL en netlist logique un outil de synthèse passe par quatre étapes :

- ✓ La première étape est la traduction du code HDL sous une spécification intermédiaire « Binary Decision Diagrams » – une méthode efficace pour représenter des équations booléennes.
- ✓ La deuxième étape effectue la transformation de la description booléenne non – optimisée convenable pour la compilation par des méthodes algorithmiques de la synthèse. Ce processus réduit chaque structure complexe dans l'application, ce qui augmente la rapidité de la synthèse, car les outils de synthèse donne des meilleurs résultats en travaillant avec des petits morceaux de la description logique.
- ✓ La troisième étape, la structuration, essaie de trouver des variables communes et de créer des structures supplémentaires dans la description logique. Il est utilisé pour diminuer la surface de la description crée par le processus précédent contenant un grand nombre des « fanout » et « fanin ».
- ✓ La quatrième étape, le « mapping » génère la netlist logique optimisée qui remplace des portes logiques dans la description avec des cellules de la bibliothèque technologique.

Le résultat de sortie est une netlist logique représenté sous le format (*.db) ou Verilog. Le format Verilog est de plus en plus utilisé comme un format intermédiaire pour transporter la netlist logique de l'outil de synthèse vers les outils de placement et de routage. Les informations temporelles sont enregistrées dans le fichier SDF (Standard Delay Format). Dans le fichier SDF on retrouve le timing des cellules individuelles de la netlist et le timing additionnel dû à l'interconnexion. Les contraintes temporelles sont sauvegardées sous le format SDC (Synopsys Design Constraint). Il est possible de générer une netlist sous le format VHDL pour la simulation post-synthèse.

5.1. Les résultats de la synthèse

Nous avons utilisé l'outil Design Compiler de Synopsys et la bibliothèque technologique HCMOS9GP, 0,13 μm de STMicroelectronics pour la synthèse de notre application.

Les résultats de la synthèse ne contiennent pas la surface du processeur ARM et des mémoires car ils étaient fournis comme des boites noires. Alors les présentes données représentent

seulement le nombre des portes logiques et la surface nécessaire pour implémenter les périphériques autour du processeur ARM.

Le sous-système VLC :

		Synthèse	
Cellules standard :	**Nombre des cellules :**	**944**	
	Nombre des interconnexions :	**1428**	
	Nombre des ports :	**22**	
	Nombre des lignes :	**21**	
Surface :	**Total combinatoire :**	**65577.54**	**µm²**
	Total séquentielle :	**2359848.25**	**µm²**
	Total des cellules	**2425435.00**	**µm²**

Le sous-système DivX :

		Synthèse	
Cellules standard :	**Nombre des cellules :**	**753**	
	Nombre des interconnexions :	**1214**	
	Nombre des ports :	**6**	
	Nombre des lignes :	**6**	
Surface :	**Total combinatoire :**	**50478.93**	**µm²**
	Total séquentielle :	**2342722.75**	**µm²**
	Total des cellules	**2343722.50**	**µm²**

Le module Splitter :

		Synthèse	
Cellules standard :	**Nombre des cellules :**	**122**	
	Nombre des interconnexions :	**319**	
	Nombre des ports :	**25**	
	Nombre des lignes :	**15**	
Surface :	**Total combinatoire :**	**58809.44**	**µm²**
	Total séquentielle :	**71067.88**	**µm²**
	Total des cellules	**129877.42**	**µm²**

Le module Combiner :

		Synthèse	
Cellules standard :	**Nombre des cellules :**	**154**	
	Nombre des interconnexions :	**357**	
	Nombre des ports :	**249**	
	Nombre des lignes :	**16**	

Surface :	**Total combinatoire :**	**17186.38**	**µm²**
	Total séquentielle :	**21743.52**	**µm²**
Total des cellules		**38929.95**	**µm²**

Le contrôleur DMA :

		Synthèse	
Cellules standard :	**Nombre des cellules :**	**351**	
	Nombre des interconnexions :	**1367**	
	Nombre des ports :	**41**	
	Nombre des lignes :	**41**	
Surface :	**Total combinatoire :**	**41567.38**	**µm²**
	Total séquentielle :	**39295.35**	**µm²**
Total des cellules		**80863.50**	**µm²**

5.2. L'estimation de la puissance consommée

Pour estimer combien de puissance consomme notre application, nous avons étudié l'architecture du sous-système VLC au niveau RTL. L'architecture de ce sous-système est présentée à la figure 5.24.

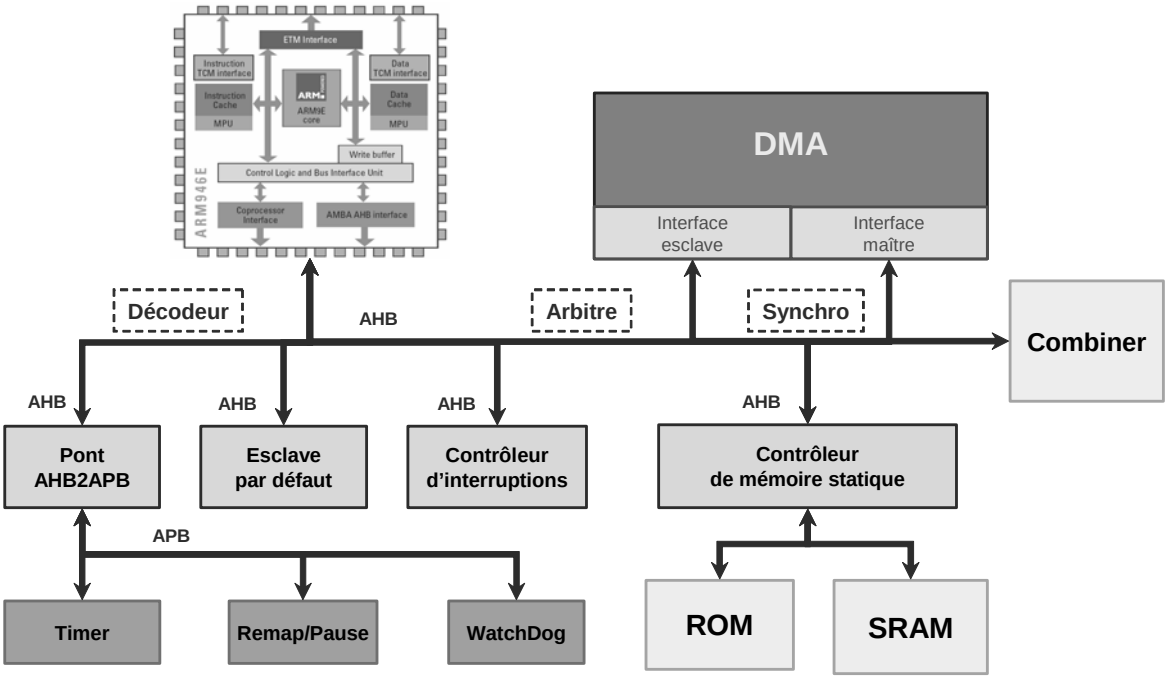


Figure 5.24 : L'architecture du sous-système VLC de l'encodeur DivX (version 2).

L'estimation de la consommation d'une telle architecture se fait en plusieurs étapes.

La première étape, consiste, une fois que le code VHDL de l'architecture a été validé par simulation au niveau RTL, à faire la synthèse des modules dont on souhaite obtenir la

consommation. Cette synthèse doit transformer les modules au niveau RTL et aux modules au niveau portes logiques. En effet, un fichier (*.db) généré après la synthèse au niveau porte logique constitue l'un des deux points d'entrée des outils d'estimation de la consommation. Le deuxième point d'entrée est un fichier des traces d'activité (*.vcd). Pour l'outil PrimePower de la société Synopsys, utilisé dans cette étude, un fichier avec les traces d'activité (*.vcd) est obtenu à partir de la simulation post-synthèse de l'architecture, c'est a dire, sur la simulation de l'architecture synthétisée (*.vhd). Cette simulation post-synthèse doit se faire sur l'architecture synthétisée au niveau porte logique. La figure 5.25 ci-dessous résume toutes ces étapes.

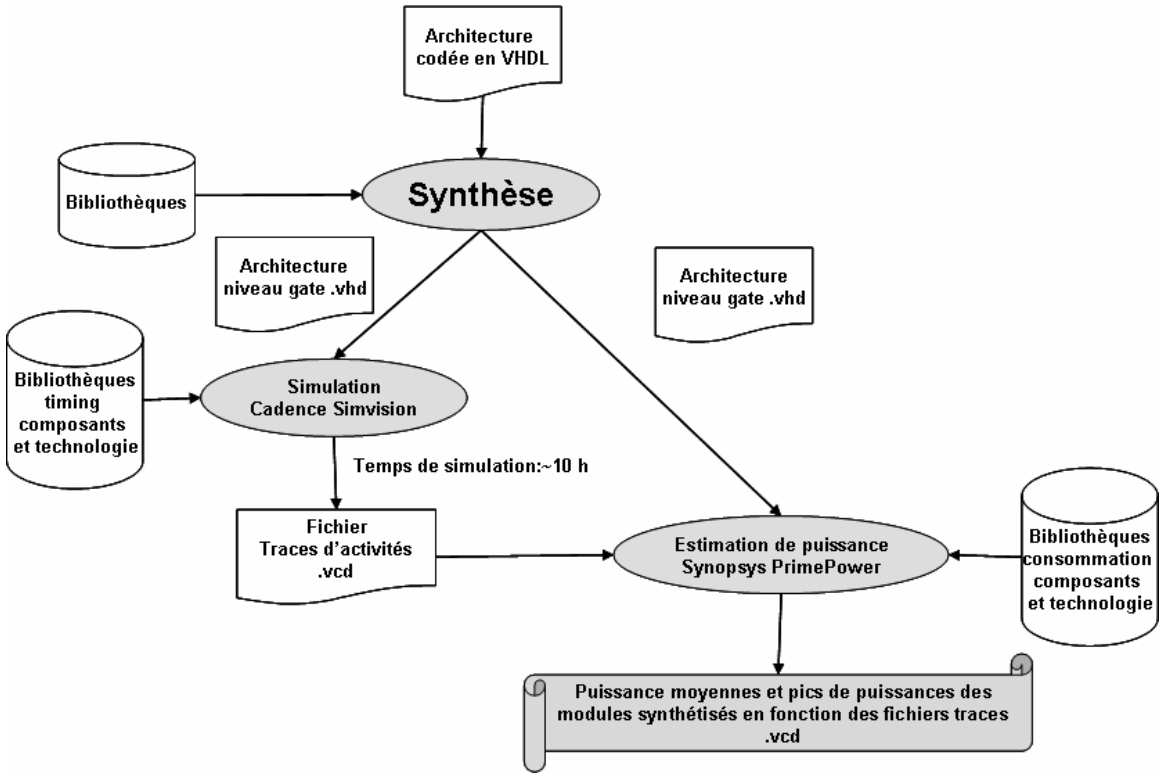


Figure 5.25 : Flot d'estimation de la consommation.

5.2.1. Les résultats de l'estimation de la puissance consommée

La synthèse a été réalisée avec une technologie HCMOS9GP 0.13 µm, alimenté à 1.2V de la société STMicroelectronics. La simulation post-synthèse, comprenant le processeur ARM946E-S prend plus qu'une journée pour simuler seulement l'encodage de la première image. Pour la génération d'un fichier traces (*.vcd) qui y fait suite cela prend environ 10 à 30 minutes (tout dépend de la période sélectionnée), et l'estimation de la puissance prend quelques minutes. La simulation post-synthèse simule environ 8 millions instructions car nous n'avons pas utilisé des mémoires caches, parce qu'elles n'étaient pas disponibles pour cette simulation. Toutefois, le rapport de l'outil PrimePower indique les puissances moyennes consommées pour chaque module synthétisé pour un période de quelques millisecondes de simulation.

Les modules dont une estimation de la consommation est donnée par PrimePower sont :

- ✓ l'arbitre,
- ✓ le décodeur d'adresses,
- ✓ les divers multiplexeurs,
- ✓ le pont liant le bus AHB et le bus APB,
- ✓ le contrôleur de la mémoire statique,
- ✓ le Default Slave,
- ✓ le périphérique « Remap/Pause »,
- ✓ le périphérique « WatchDog »,
- ✓ le contrôleur « Timer »,
- ✓ le contrôleur d'interruptions,
- ✓ le contrôleur de reset,
- ✓ le module Combiner, propre à l'application du VLC.

Voyons maintenant quelle est la part de chacun de ces modules dans la consommation de l'architecture totale hors processeur et mémoires. Pourcentage de la contribution en puissance dynamique des modules synthétisés :

- ✓ le module DefaultSlave : 0.0002%,
- ✓ le contrôleur de Reset : 0.6%
- ✓ le module Synchro : 0.6%
- ✓ le module Remap/Pause : 1.18%
- ✓ le module WatchDog : 3.6%
- ✓ le module Timer : 14.2%
- ✓ le contrôleur d'Interruptions : 29%
- ✓ le module Combiner, propre à l'application du VLC : 41%
- ✓ les Interconnexions : 11 %

Un commentaire est nécessaire pour expliquer le pourcentage de la contribution en puissance du module Combiner. Cette consommation de puissance est dûe au bloc mémoire intégré dans la FIFO du module Combiner qui est accédé très fréquemment. La mémoire implémentée est une mémoire statique synchrone double port de 16 mots de 32 bits.

5.2.2. La consommation des mémoires présentes dans l'architecture :

La même démarche que le travail présenté ci-dessus n'a pas pu être effectuée pour l'estimation de la puissance consommée par les mémoires. En effet, il n'y a pas besoin de les synthétiser et d'estimer leur consommation à l'aide d'outil PrimePower de Synopsys.

Toutefois, les mémoires implémentées sont des mémoires statiques et leur comportement énergétique est déjà étudié et ne présente aucune particularité. Disposant de la documentation des mémoires SSRAM, nous avons pu donner une estimation de leur contribution lors de chaque accès.

Les valeurs suivantes sont extraites pour une mémoire SRAM de 4096 mots x 32 bits, synchrones et de single port, avec 8 multiplexeurs. Elles proviennent de la technologie HCMOS9GP, avec 6 transistors par cellule de mémoire (de taille 2.5064 μm² pour chacune).

Ainsi, la documentation nous indique que la consommation pour chaque type d'accès est de :

- ✓ Accès en écriture : 75 μW /MHz, soit à 66MHz, 4908,7μW ;
- ✓ Accès en lecture : 62,5 μW /MHz, soit à 66MHz 4131,2 μW

La durée de tels accès est approximativement de 2,2 ns.

Alors, les énergies à rajouter pour chaque type d'accès dans notre application sont de l'ordre de $1.08.10^{-11}$ J pour l'écriture et de 9.10^{-12} J pour la lecture. Et la puissance consommée lorsque aucun accès à la mémoire n'est effectué est de 209.9 μW.

6. Conclusion

Dans ce chapitre, nous avons montré le développement de l'architecture de l'encodeur DivX. Nous avons présenté aussi l'environnement de covérification utilisé pour valider notre système monopuce multiprocesseurs qui était adapté pour correspondre aux conditions de la flexibilité et de la testabilité dans la phase prototypage d'un système monopuce.

Grâce à cette plateforme de vérification, nous avons pu développer simultanément le matériel et le logiciel de notre système. En conséquence, la séparation de la phase de vérification dans deux étapes a simplifié le processus global de la validation et il a accéléré le débogage de l'application DivX. Nous avons présenté aussi les résultats de la simulation, de la synthèse et de l'estimation de la puissance de l'architecture de l'encodeur DivX.

Chapitre 6

CONCLUSIONS ET PERSPECTIVES

1. Le bilan de la thèse

Ce chapitre conclut la thèse en donnant un bilan du travail effectué et les perspectives envisageables au terme de cette recherche. Le cadre de mes travaux a toujours été la conception physique des systèmes monopuce multiprocesseur. J'ai eu la chance de pouvoir appliquer en pratique tout le flot de conception à partir de la conception du logiciel embarqué de bas niveau pour un processeur ARM jusqu'au placement et routage de la topologie des interfaces de communication de l'application VDSL. Même si j'ai eu cette possibilité, ma thèse était dirigée essentiellement dans deux directions de recherche : la conception des descriptions efficaces des composants matériels à partir d'un niveau d'abstraction plus élevé en utilisant des outils de raffinement automatique, et le passage du modèle de simulation d'une application de haut niveau vers le prototypage de cette application pour rendre utiles les outils de conception développés.

Je voudrais rappeler les contributions apportées par ces travaux de thèse du point de vue scientifique. Mais avant de le faire, je tiens à faire apparaître la contribution que cette thèse a apportée à moi-même. Cette contribution importante n'était pas séparée et formalisée dans un seul chapitre, mais j'ai essayé de la démontrer dans tous les exemples et parfois commentaires de ce mémoire de thèse. Elle représente la tâche fastidieuse de la conception des systèmes aux différents niveaux d'abstraction. Cette tâche a nécessité d'acquérir plusieurs types de connaissances approfondies de la manière de décrire les circuits à chaque niveau d'abstraction. Elle a nécessité d'adopter les différentes point de vue de chaque groupe de concepteurs sur la façon dont ils explorent les caractéristiques d'une application jusqu'aux détails techniques des outils utilisés dans le processus de la conception et de la simulation d'un circuit. Maintenant voyons les problèmes et les contributions abordés par cette thèse.

Pendant la première expérimentation, la conception du layout du modem VDSL [PET03], traitée en partie au laboratoire ECAD de Sofia, mon travail de thèse avaient fait apparaître la nécessite de bien prendre en compte la technique de codage des composants matériels dans le processus de génération automatique des descriptions au niveau RTL. A l'issue des problèmes rencontrés avec cette application, nous avons pu formaliser l'implémentation et la réalisation physique des composants matériels décrits en SystemC pour des systèmes sur puce multiprocesseur. Cette formalisation était présentée sous forme de proposition de quelques règles de codage pour éviter les problèmes qui peuvent apparaître pendant l'étape de synthèse pour les descriptions décrites en SystemC au niveau RTL générées automatiquement par des outils de raffinement.

Cette formalisation d'implémentation des composants matériels est encore plus utile avec le développement de l'encodeur DivX, où nous avons voulu nous rapprocher de la vraie problématique de la conception des systèmes monopuce multiprocesseur. Ce sont les problèmes que l'industrie électronique rencontre avec les applications à la pointe dans le domaine du traitement des données multimédia.

Une grande partie de mes travaux de thèse était consacrée aux détails d'intégration d'une architecture au niveau RTL à partir de l'architecture au niveau système d'une application d'encodage vidéo en temps réel. Le problème de conception de chacun des sous-systèmes de cette application, implémentés autour d'un processeur ARM946E-S, a été concentré sur le développement d'une plateforme appropriée de vérification au niveau RTL pour fusionner le logiciel et le matériel de l'encodeur DivX dans un prototype correspondant aux conditions du modèle de simulation et des spécifications du système de haut niveau. Nous avons voulu créer un prototype au niveau RTL qui sera le plus proche possible d'un circuit réel en utilisant tous les moyens existants pour imiter le vrai fonctionnement des composants comme par exemple l'annotation des diagrammes de temps et de retards etc. Ce travail a été présenté dans l'article [PET05] à la conférence MIXDES 2005.

L'expérimentation avec le prototypage de l'application de l'encodage DivX a mis en évidence certaines difficultés concernant l'intégration du logiciel et du matériel dans un prototype physique. Les travaux menés ont défini la deuxième contribution principale de cette thèse, la méthode pour faciliter l'intégration du matériel et du logiciel à partir d'un modèle au niveau de bus fonctionnel, qui est publié dans l'article [PET05a] à la conférence RSP 2005. Cette méthode vise à réduire le temps et les efforts pendant le développement du prototype d'un système monopuce, nécessaires pour produire le code de logiciel considérant les détails de l'architecture de plus bas niveau, tels que la configuration et l'initialisation du processeur, ainsi que la configuration et le chargement de la mémoire. Nous avons mis en application des techniques pour contourner ces difficultés, mais il reste encore à développer des outils pour faciliter cette phase de prototypage.

2. Les perspectives

Les perspectives envisageables en prolongement direct de cette thèse concernent deux objectifs importants : automatisation du passage du niveau de bus fonctionnel vers le niveau RTL et l'augmentation de l'abstraction et de la flexibilité dans l'interfaçage de matériel et de logiciel.

L'automatisation de l'intégration du logiciel et de matériel à partir de niveau de bus fonctionnel vers le niveau RTL. Nous sommes fortement motivés par les travaux liés avec l'automatisation du passage entre les deux niveaux. Nous pensons que c'est un axe de recherche

qui nécessite d'être exploré. En effet nous avons déjà commencé à étudier cette perspective, en nous intéressant à la façon dont l'intégration du logiciel et de matériel développé peut être réalisée dans le flot de conception existant du groupe SLS. Ces travaux de recherche feront partie de la thèse de Marcio Oyamada.

Plus d'abstraction et de flexibilité dans l'interfacage de matériel et de logiciel. L'étude sur la conception et le prototypage de l'architecture de l'encodeur DivX au niveau RTL a établi le cadre conceptuel de ce travail. Ce travail va servir comme un point de référence pour les travaux qui vont poursuivre la conception de l'application en essayant d'augmenter l'abstraction de l'interface matériel/logiciel. Nous pensons qu'il est important de poursuivre cette étude en pensant à remplacer le processeur ARM avec un processeur flexible (comme par exemple le processeur XTENSA de Tensilica). Cette approche va obligatoirement provoquer plusieurs changements dans l'architecture de chaque sous-système, ce qui représente un vrai défi pour nous et pour les travaux effectués.

Bibliographie

[AMB02] ARM Ltd., *"AMBA AHB-Lite Overview"*, disponible à www.arm.com, 2002.

[AND04] J. Andrews, *"ARM SoC Verification Matrix Improves HW/SW Co-Verification"*, Electronic Design Processes 2004 April 25-27, 2004, Monterey, CA

[ARM98] ARM Ltd, *"Application Note 48: Scatter loading"*, disponible à www.arm.com, 1998.

[ARM01] ARM Ltd, *"ARM Developer Suite 1.2"*, disponible à www.arm.com, 2001.

[ARM02] ARM Ltd, *"AMBA Multi-layer AHB Overview"*, disponible à www.arm.com, 2002.

[ARM02a] ARM Ltd, *"Application Note 107: Embedded Software Development with ADS 1.2"*, disponible à www.arm.com, 2002.

[ARM03] ARM Ltd., *"ARM946E-S Technical Reference Manual"*, disponible à www.arm.com, 2003.

[ARM04] ARM Ltd., "*MaxSim Developer Suite User's Guide*", AXYS Documentation, 2004

[ARM99] ARM Ltd., *"Advanced Microcontroller Bus Architecture Specification"*, disponible à www.arm.com, 1999.

[ART03] A. Artieri, V. D'Alto, R. Chesson, M. Hopkins, M.C. Rossi, "*Nomadik™ Open Multimedia Platform for Next-generation Mobile Devices*", STMicroelectronics Technical Article TA305, 2003, disponible à www.st.com.

[BAG01] A. Baghdadi, *"Exploration et conception systématique d'architecture multiprocesseurs monopuces dédiées à des applications spécifiques"*, Thèse de Doctorat INPG, laboratoire TIMA, May 2002.

[BRI03] S. Brini, D. Benjelloun, F. Castanier, *"A Flexible Virtual Platform for Computational and Communication Architecture Exploration of DMT VDSL Modems"*, Proceedings of the Design Automation and Test in Europe (DATE), Munich, Germany, March 3-7, 2003.

[BUD00] M. Budagavi, W. Rabiner, J. Webb, R. Talluri, *"Wireless MPEG-4 video communication on DSP chips"*, Signal Processing Magazine, IEEE Publication Date: Jan 2000 Volume: 17, Issue: 1

[CAD04] Cadence Design Systems, *"Cadence NC-Sim Simulator"*, product documentation, 2004.

[CAD05] Cadence Design Systems, *"Cadence Incisive Platform"*, product datasheet, 2005.

[CAI03] L. Cai, D. Gajski, *"Transaction Level Modeling: An Overview"*, CODES+ISSS'03, Newport Beach, USA, October 1–3, 2003.

[CES99] W.O. Cesario, *"Synthèse Architecturale Flexible "*, Thèse de doctorat, INPG, laboratoire TIMA, 1995.

[CES02] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, A.A. Jerraya, *"Multiprocessor SoC Platforms: A Component-Based Design Approach"*, IEEE Design & Test of Computers, Vol. 19 No. 6, November/December 2002.

[COM02] Comelec, *"Spécification et création d'une bibliothèque de blocs IP"*, disponible à www.comelec.enst.fr, 2002.

[COW04] CoWare Inc., "ConvergenSC/Incisive Design Flow White Paper", disponible à www.coware.com, June 2004

[DDI04] ARM DDI 0196F, *"PrimeCell DMA Controller (PL080) : Technical Reference Manual"*, disponible à www.arm.com, 2004

[DSD04] A.A. Jerraya, *"Long Term Trends for Embedded System Design"*, Euromicro Symposium on Digital System Design (DSD 2004), Rennes, France, Sept. 2004.

[DUT01] S. Dutta, R. Jensen, and A. Rieckmann, *"Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems,"* IEEE Design and Test of Computers, Vol. 18 No. 5, September/October 2001.

[ETZ03] K. Etzel, *"Quelle architecture SRAM choisir pour assurer une performance maximum"*, EETimes, France, Novembre 2003.

[GAJ00] D. Gajski et al., *"SpecC: Specification Language and Methodology"*, Kluwer, 2000.

[GAU02] L. Gauthier, *"Génération de systèmes d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseur hétérogènes dans le cadre des systèmes embarqué spécifiques"*, Thèse de doctorat, INPG, laboratoire TIMA, 2001.

[GEB05] B. Gebremichael, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, A. Radulescu, *"Deadlock Prevention in the ÆTHEREAL Protocol"* Proceeding of the CHARME 2005, Saarbrücken, Germany, 3-6 October 2005

[GHA03] F. Gharsalli, *"Conception des interfaces logiciel/matériel pour l'intégration des mémoires globales dans les systèmes monopuces"*, Thèse de doctorat, INPG, laboratoire TIMA, 2003.

[GHO95] A. Ghosh, M. Bershteyn, et al., "A Hardware-Software Co-Simulation for Embedded Systems Design and Debugging", Proceedings of the Conference on Asia Pacific Design Automation, Makuhari, Massa, Chiba, Japan, August 29 - September 1, 1995.

[GRA04] A. Grasset, F. Rousseau, A. A. Jerraya, *"Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation"*, 15th IEEE International Workshop on Rapid System Prototyping (RSP 2004), Geneva, Switzerland, June 2004.

[HAR03] P. Hardee, *"Transaction-level Modeling and the ConvergenSC Products"*, CoWare Inc., www.coware.com, 2003.

[HAV02] A. Haverinen, M. Leclercq, N. Weyrich, and D. Wingard, *"SystemC™ based SoC Communication Modeling for the OCP™ Protocol"*, OSCI Technical Paper, October at www.systemc.org, 2002.

[HEL02] J. Helmig, *"Developing core software technologies for TI's OMAPTM platform,"* Texas Instruments, 2002. disponible à www.ti.com.

[HIT97] Hitachi Inc, *"Hitachi to realizes the HG73M series of ASIC that integrate High Speed Logic and a High Density DRAM on the same chip"*, disponible à www.hitachi.com, 1997.

[IEE88] IEEE Standard 1076-1987, *"IEEE Standard VHDL Language Reference Manual"*, IEEE Press, 1988.

[IEE96] IEEE Standard 1364-1995, *"IEEE Hardware Description Language Based on the Verilog"*, IEEE Press, 1996.

[INT93] Intel Corp., *"8237A High Performance Programmable DMA Controller"*, datasheet documentation, September 1993

[IPC] *"Inter-Process Communication Tutorial"* disponible à www.cne.gmu.edu/modules/ipc/

[IEE01] IEEE Standard for VITAL ASIC (Application Specific Integrated Circuit) Modeling Specification, IEEE Std 1076.4-2000, IEEE Press, Ref. SH94959, 2001

[JER04] A.A. Jerraya, *"Spécification et validation des systèmes monopuce"* Hermes Science Publication, 2004

[JER05] A.A. Jerraya, W. Wolf. *"Hardware/Software Interface Codesign for Embedded Systems,"* IEEE Computer, Vol. 38, No. 2, February 2005.

[KAR02] F. Karim, A. Nguyen, S. Dey, *"An Interconnect Architecture for Networking Systems on Chips"*, IEEE Micro, Vol. 22, No. 5, September/October 2002.

[KEU00] K. Keutzer, S. Malik, R. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, *"System Level Design: Orthogonalization of Concerns and Platform-Based Design"*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, Vol. 19, No. 12, December 2000.

[KIM96] K. Kim, Y. Kim, Y. Shin, K. Choi, *"An Integrated Hardware/Software Cosimulation Environment with Automated Interface Generation"*, in proc of 17th IEEE International Workshop on Rapid Prototyping, pages 66-71, June 1996.

[KOE02] R. Koenen *"Overview of the MPEG-4 Standard"*, disponible à http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm, 2002

[LEN00] C. Lennard, D. Lin, K. Ho, and V. Chandran, *"Collaborative Development of Bluetooth IP: Fundamentals of System-Modeling in Complex IP Creation and Reuse"*, in proc. of IP 2000 Bluetooth Conference, October 2000.

[LEN05] C. Lennard, D. Mista, "*Taking Design to the System Level"*, ARM Press disponible à www.arm.com, April 2005.

[LYO03] D. Lyonnard, *"Approche d'assemblage systématique d'élément d'interface pour la génération d'architecture multiprocesseurs"*, Thèse de doctorat, INPG, laboratoire TIMA, 2003.

[MAY03] OpenDivX, *"Project Mayo"*, disponible à http://www.projectmayo.com, 2003.

[MEN03] Mentor Graphics, *"Seamless Hardware/Software Co-Verification"*, product datasheet at www. mentor.com/seamless, 2003.

[MEN04] Mentor Graphics, *"Platform Express User's Guide"*, at www. mentor.com, 2004.

[MPI95] "The Message Passing Interface (MPI) standard", http://www-unix.mcs.anl.gov/mpi/, Version 1.1, 1995

[NAC99] F. Nacabal, C. Valderama, F. Hessel, P. Paulin, A.A. Jerraya, *"Cosimulation C-VHDL pour la validation fonctionnelle de logiciel embarqué"*, Technique et science informatiques, Vol. 19 No. 8, Hermes Science Publications, October 2000.

[NIC01] G. Nicolescu, S. Yoo, A.A. Jerraya, "*Mixed-Level Cosimulation for Fine Gradual Refinement of Communication is SoC Design"* in proc. Design Automation and Test in Europe, Munich, Germany, March 13-16 2001.

[NIC02] G. Nicolescu, *"Spécification et validation des système hétérogènes embarqués"*, Thèse de doctorat, INPG, laboratoire TIMA, 2002

[NIC02a] G. Nicolescu, K. Svarstad, W. Cesario, L. Gauthier, D. Lyonnard, S. Yoo, P. Coste, A.A. Jerraya, *"Desiderata pour la spécification et la conception des systèmes électroniques"*, Technique et Science Informatiques, March 2002.

[OC04] Open Cores, *"SoC Interconnection : Wishbone"*, disponible à www.opencores.org, 2004.

[OCP04] OCP International Partnership, *"Open Core Protocol International Partnership"*, disponible à www.ocpip.org, 2004.

[PAV02] Y. Paviot, W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicoulescu, S. Yoo, A.A. Jerraya, M. Diaz-Nava,*"HW/SW Interfaces Design of a VDSL Modem using Automatic Refinement of a Virtual Architecture Specification into a Multiprocessor SoC: a Case Study"*, Proceedings of DATE 2002, Paris, France, March 2002.

[PAV04] Y. Paviot, *"Implémentations mixtes logicielles/matérielles des services de communication pour l'exploration du partitionnement logiciel/matériel"*, Thèse de doctorat, INPG, laboratoire TIMA, 2004.

[PET03] I. Petkov, P. Amblard, M. Hristov, A.A. Jerraya, *"Physical Design of HW Interfaces for MPSOC"*, 7th International MTM Symposium, Sofia-Sozopol, Bulgaria, September 2003.

[PET05] I. Petkov, P. Amblard, M. Hristov, A.A. Jerraya, *"Effective Hardware Verification of ARM Based System on Chip Design"*, 12th International Conference Mixed Design of Integrated Circuits and Systems, Krakow, Poland, 22-25 June 2005.

[PET05a] I. Petkov, P. Amblard, M. Hristov, A.A. Jerraya, "*Systematic Design Flow For Fast Hardware/Software Prototype Generation From Bus Functional Model For MPSoC"*, 16th IEEE International Workshop on Rapid System Prototyping 2005, Montréal, Canada, 8-10 June 2005.

[PTO02] Ptolemy project, disponible à https://prolemy.eecs.berkeley.edu, 2002.

[RAM03] Rambus Inc, Documentation disponible à www.rambus.com, 2003.

[ROW94] J. Rowson, *"Hardware/Software Cosimulation"*, in proc. Design Automation Conference, San Diego, California, USA, June 6-10, 1994.

[SAS03] A. Sasongko, A. Baghdadi, F. Rousseau, A.A. Jerraya, *"SoC Validation trough Prototyping on Arm Integrator Platform"*, Information Quarterly Journal, ARM Press, 2003.

[SAS04] A. Sasongko, *"Prototypage base sur une plateforme reconfigurable pour la vérification des systèmes monopuces"*, Thèse de doctorat INPG, laboratoire TIMA, 2004.

[SDF] *"Standard Delay File Format Manual"*, version 2.1 or 3.0, Open Verilog International, disponible à http://www.ovi.org/pubs.html.

[SIL04] Silicore Corporation, *"The WISHBONE Service Center"*, disponible à www.silicore.net, 2004

[SON02] Sony Corporation, *"Toshiba and Sony make major advance in semiconductors process technologies"* disponible à www.sony.net, 2002.

[SPH02] SPHS9GP Memory Product Information, 2002

[SPI00] P. Guerrier, A. Grenier, *"A Generic Architecture for On-Chip Packet –Switched Interconnections"*, in proc. Design Automation and Test in Europe, Paris, France, March 4-8, 2002.

[SUT02] S. Sutherland, *"Verilog 2001 - A Guide to New Features of the Verilog Hardware Description Language*", Springer, January 2002.

[SWP01] SWPY001 *"OMAP Technology Overview White Paper"*, disponible à www.ti.com, 2002

[SYN02] Synopsys Inc., *"Describing Synthesizable RTL in SystemC™ 1.1"*, disponible à www.synopsys.com, 2002.

[SYN03] Synopsys Inc, "*System Studio*", product datasheet at www.synopsys.com, 2003.

[SYN04] Synopsys Inc, *"Synopsys coreTools for IP Reuse"*, product datasheet at www.synopsys.com, 2003.

[SYN05] Synopsys Inc, "*SmartModel Library: Release notes*" disponible à www.synopsys.com, 2005

[SYS03] Open SystemC Initiative, *"SystemC 2.0.1 Language Reference Manual"*, disponible à www.systemc.org, 2003.

[TCL05] *"TCL Reference Manual"* disponible à http://tmml.sourceforge.net/doc/tcl/, 2005.

[TEN02] Tensilica Inc., *"XTENSA: Architecture and performance"*, disponible à www.tensilica.com, 2002.

[VCC03] VCC, *"Cadence Virtual Component Co-design Modeling guide,"* disponible à www.cadence.com/products/vcc.html

[VER03] SystemVerilog 3.1, "Accellera's Extensions to Verilog", Accelera Web Press, 2003

[VIN02] A. S. Vincentelli, *"Defining Platform-based Design"*, EEDesign of EETimes, February 2002.

[VSI04] VSI Alliance, "*VSI Alliance Specifications*", disponible à www.vsi.org, 2004.

[WAG04] F.R. Wagner, L. Carro, W.O. Cesario, A.A. Jerraya, "*Strategies for the Integration of Hardware and Software IP Components in Embedded Systems-on-Chip*", Integration - the VLSI Journal, Elsevier, Vol. 37, Issue 4, September 2004.

[WAS04] M.-W. Youssef, S. Yoo, A. Sasongko, Y. Paviot, A.A. Jerraya, "*Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study*", DAC'04, San Diego, USA, June 2004.

[KLE04] R. Klein, K. Travilla, M. Lyons, *"Performance Estimation of MPEG4 Algorithms on ARM Architectures Using Co-Verification"* Embedded System Conference'04, San Francisco, USA, 2004

TITRE : CONCEPTION DES SYSTEMES MONOPUCE MULTIPROCESSEUR : DE LA SIMULATION VERS LA REALISATION

RESUME :

La complexité des systèmes monopuce est devenue telle qu'il est impossible de continuer à les concevoir au niveau RTL, où il faut préciser chaque détail du comportement des composants. Le grand défi en ce moment pour les ingénieurs est de réussir à maîtriser la complexité lors de la conception de ces systèmes et d'arriver à une conception rapide des systèmes monopuce sous de fortes contraintes de qualité et de temps de développement. Pour dépasser ce défi, les nouvelles méthodes de conception sont basées sur des concepts d'abstraction de haut niveau. La problématique de cette thèse est de comprendre les difficultés de la conception des systèmes sur puce commençant à un niveau d'abstraction élevé et d'essayer de trouver des méthodes ou techniques pour faciliter et accélérer leur développement. Nous nous sommes posé comme objectif d'étudier différentes méthodologies de prototypage des systèmes monopuce et les problèmes liés avec les niveaux d'abstraction et les outils de conception. Les contributions apportées par cette thèse, trouvent place dans la conception des systèmes multiprocesseurs hétérogènes à l'étape d'intégration de matériel et de logiciel à partir d'un modèle abstrait et dans le prototypage des applications monopuce multiprocesseur

TITLE: DESIGN OF MULTIPROCESSOR SYSTEM ON CHIP: LINK BETWEEN SIMULATION AND REALIZATION

ABSTRACT:

The design of the system on chip at RTL level is no longer practical approach due to the rising complexity of the circuits. It became difficult to specify all details of the behavior of each component and to validate the entire system at RTL level. The challenge in our days for the engineers is to succeed to control the complexity when designing these systems and to speed up the design of the systems under strong quality constraints and time to market pressure. To deal with this challenge, the new methods of design are based on concepts of high-level abstraction of the system components. The goals of this thesis are to understand the difficulties of the systems on chip design starting on a high level of abstraction and to try to find methods or techniques to accelerate their development. We focused our objectives to study different methodologies for prototyping of system on chip and to study the different types of components used at different levels of abstraction and the relations between them during prototyping. The contributions of this thesis, find place in the design of heterogeneous multiprocessors systems on chip at the stage of integration of hardware and software starting from an abstract model to the prototyping of the multiprocessor system on chip.

ЗАГЛАВИЕ: ПРОЕКТИРАНЕ НА СИСТЕМИ ВЪРХУ ЧИП: ОТ СИМУЛИРАНЕ КЪМ РЕАЛИЗАЦИЯ

АБСТРАКТ:

Нарастващата сложност на системите върху чип накара да се потърсят нови технологии за проектиране на системи върху чип, различни от тези познати ни досега на ниво RTL. Това се дължи на невъзможността да се опишат с подробности всички детайли от поведението на всеки компонент на системата и да се провери тяхната функционалност и тази на цялата система на ниво RTL. Предизвикателството в наши дни пред технологии за проектиране на такива схеми е да успеят да овладеят сложността по време на проектиране на системите върху чип и да ускорят процеса на тяхното проектиране пред все по-високи изисквания за качество и пред все по-силен натиск на пазара. За да се справят с това предизвикателство, новите методологии на проектиране използват методи позволяващи описание на системите на още по-високо ниво на абстракция от RTL, наречено системно ниво. Целта на тази дисертация е да изследва и разбере методологиите за проектиране на системи върху чип на ниво система и да предложи техники и методи с които цели да ускори и улесни тяхното проектиране при преминаване от по-високо ниво на проектиране към по-ниско. За да се постигне поставената задача, насоката на представена научна работа беше фокусирана върху изучаването на различените методологии за прототипиране на системите върху чип и към изследването на различията и взаимовръзката между отделните компоненти съставящи една такава система на всяко едно от различните нива на описание на схемата по време на проектиране. Приносите на тази дисертация са приложими в областта на проектиране на многопроцесорни системи върху чип в процеса на интегриране на отделните части на системата.

INTITULE ET ADRESSE DES LABORATOIRES

Laboratoire TIMA, 46 avenue Félix Viallet, 38031Grenoble Cedex, France
Laboratoire ECAD, 8 boulevard Kliment Ohridksi, 1000 Sofia, Bulgarie

ISBN : 2-84813-082-2 **ISBNE : 2-84813-082-2**