Conception de haut niveau des MPSoCs à partir d'une spécification Simulink: Passerelle entre la conception d'algorithmes et la conception d'architectures

Y. Atat

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|

THESE

Pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

Préparée au laboratoire TIMA dans le cadre de
L'Ecole Doctorale Electronique Electrotechnique Automatique Télécommunications Signal

Présentée et soutenue publiquement

Par

Youssef ATAT

Le 21 Mai 2007

Titre :

Conception de haut niveau des MPSoCs à partir d'une spécification Simulink : Passerelle entre la conception au niveau Système et la génération d'architecture

Directeur de thèse : Ahmed- Amine JERRAYA
Co-directeur: Nacer-Eddine ZERGAINOH

JURY

M. Pierre GENTIL	**, Président**
M.Abbas DANDACHE	**, Rapporteur**
M. Jean-Philippe DIGUET	**, Rapporteur**
M. Ahmed-Amine JERRAYA	**, Directeur de thèse**
M. Nacer -Eddine ZERGAINOH	**, Codirecteur**
M. Olivier SENTIEYS	**, Examinateur**

Remerciement

Au terme de ce travail de thèse et de cette épisode de ma vie je tiens à exprimer mes remerciements à toutes les personnes qui ont contribué à accomplir ce travail. Je remercie :

Monsieur Ahmed Amine Jerraya et Monsieur Nacer-eddine Zergainoh qui m'ont encadré et qui m'ont dirigé durant cette thèse.

Monsieur Pierre Gentil, directeur de l'école doctorale EEATS d'avoir presidé mon jury de thèse et de m'avoir soutenu et donné des conseils précieux durant ces années de thèse.

Mes deux rapporteurs Monsieur Abbas Dandache, professeur de l'université de Metz et Monsieur Jean-Philippe Diguet, chargé de recherche CNRS à l'université de bretagne sud pour l'étude et l'analyse approfondies de mon travail.

M. Olivier Sentieys, professeur de l'université de Rennes I pour sa disponibilité et ses remarques judicieux.

A mon père, ma mère

Bilal, Ahlam

& Zeinab

Table des matières

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

INTRODUCTION GENERALE

Sommaire

Les systèmes sur puce ou systèmes monopuces (SoC, pour System-On-Chip) deviennent de plus en plus complexe. Cette complexité croissante est accentuée par l'émergence de nouvelles applications télécoms et multimédia avec des contraintes fonctionnelles de plus en plus sévères (puissance de calcul, consommation, d'embarquabilité, reconfigurabilité) et non fonctionnelles (temps de mise sur le marché, rétrécissement de la durée de vie du produit, coût, …). Pour répondre à ces exigences et maîtriser cette complexité de nombreux travaux de recherche et de développement sont menés aussi bien sur les aspects architecturaux et technologiques que sur les aspects méthodologies et d'outils de conception. Concernant les architectures et technologie de fabrication, l'intégration de dizaines, voir des centaines de cœurs de traitement autour d'un réseau de communication sophistiqué sur une seule puce domine les axes de recherche actuels pour répondre aux contraintes de ces nouvelles applications télécoms et multimédia. Il s'agit donc d'une évolution des architectures des SoCs vers des architectures multiprocesseurs, c'est-à-dire, des systèmes multiprocesseurs sur puce (MPSoC, pour Multi-Processor System-On-Chip) et réseaux sur puce (NoC, pour Network-On-Chip). Concernant les méthodologies et les outils de conception, le défi est de maîtriser la complexité en remontant le niveau d'abstraction au plus haut niveau et en automatisant le flot du passage de la spécification algorithmique à l'implémentation validé et l'intégration du système complet matériel/logiciel.

C'est dans le contexte de conception de haut niveau des systèmes multiprocesseurs monopuces que notre travail souhaite apporter une contribution qu'il nous faut tout d'abord délimiter.

1. Système multiprocesseur monopuce MPSoC

Un système multiprocesseur sur puce MPSoC est un système complet intégrant sur une seule pièce de silicium plusieurs composants complexes et hétérogènes tels que des unités de calcul spécifiques programmables et/ou non programmable (CPU, DSP, ASIC-IP, FPGA) des réseaux de communication complexes (Bus hiérarchiques sur puce, réseau sur puce) des composants de mémorisation variés, périphériques E/S, etc. La Figure 1.1 représente un modèle générique d'un système MPSoC hétérogène avec des parties

matérielles et logicielles structurées en couches pour maîtriser la complexité. Le matériel se divise en deux couches :

- La couche basse contient les composants de calcul et de mémorisation utilisés par le système (µP, µC, DSP, matériel dédié IPs, mémoires) ;
- La couche matérielle de communication embarquée sur la puce composée de deux sous-couches : média de communication (liens point-à-point, bus hiérarchique, réseau sur puce) et adaptateurs de communication entre le réseau et les composants de la première couche.

Le logiciel embarqué est aussi découpé en couches :

- La couche la plus basse est l'abstraction du matériel (HAL, pour Hardware Abstraction Layer en anglais) permet de faire le lien avec le matériel en implémentant les pilotes E/S des périphériques, des contrôleurs de composants, les routines d'interruption (ISR, pour Interrupt Service Routine).
- La couche système d'exploitation qui permet de porter l'application sur l'architecture (gestion de ressources, communication et synchronisation, ordonnancement).
- La couche application.

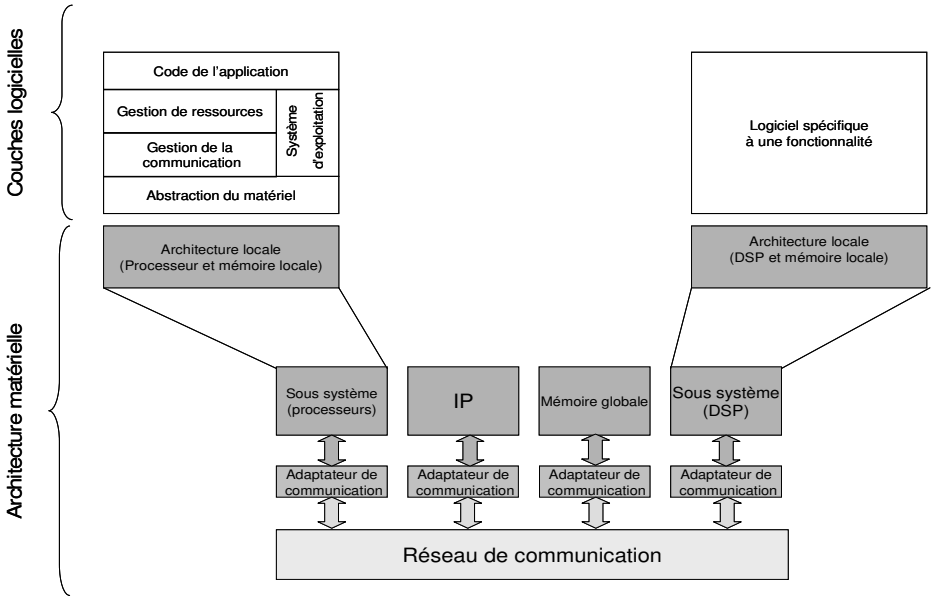


Figure 1.1: Architecture typique d'un système multiprocesseur hétérogène

2. Conception de haut niveau des MPSoCs

La conception des MPSoCs à l'aide de méthodes classiques conduit à des coûts et des délais de réalisation inacceptables. En fait, la conception de ces systèmes induit plusieurs points de cassures qui rendent difficile la mise en échelle des méthodes de conception existantes. Les principaux défis sont :

— La conception des systèmes multiprocesseurs monopuces entraînera inévitablement la réutilisation de composants existants qui n'ont pas été conçus spécialement pour être assemblés sur la même puce. Il sera donc nécessaire d'utiliser des méthodes de composition et d'assemblage de composants hétérogènes;

— Le concepteur d'un tel système contenant des processeurs doit obligatoirement fournir, en plus du matériel, une couche logicielle permettant la programmation du système intégré. Il sera donc nécessaire d'aider les concepteurs des architectures MPSoC à concevoir les couches logicielles;

— L'assemblage de composants hétérogènes sur une même puce pour concevoir des architectures dédiées à des applications particulières peut nécessiter des schémas de communication très sophistiqués. La conception de ces interfaces hétérogènes sera vraisemblablement le principal goulet d'étranglement du processus de conception des systèmes multiprocesseurs monopuces;

— La complexité des systèmes MPSoC est telle qu'il devient quasi impossible de spécifier/valider manuellement à des niveaux bas tels que le niveau transfert de registre (RTL) où il faut donner/valider des détails au niveau cycle d'horloge. Des méthodes de conception/validation basées sur des concepts de plus haut niveau sont donc nécessaires. Comme pour le logiciel, des outils de synthèse de haut niveau seront nécessaire pour le matériel. La vérification des composants synthétisés sera obligatoire afin de pouvoir en assurer la qualité.

Ainsi pour maîtriser cette complexité et répondre aux contraintes de temps de mise sur le marché et de coût, la conception des systèmes multiprocesseurs monopuces doit se baser sur des approches de conception mixtes (descendante et ascendante). C'est-à-dire aussi bien sur l'assemblage et la réutilisation de composants existants préconçus et

prévalidés et d'autre part sur la modélisation système et l'augmentation du niveau d'abstraction. Cependant, ceci va engendrer plusieurs problèmes lors de la démarche de conception des MPSoCs. Dans le cadre de cette thèse nous nous focalisons sur trois problèmes :

♦ Le premier problème est relatif à la spécification système. Pour modéliser un système, il faut d'abord faire des choix pertinents : environnements et langages de programmation, les modèles de calcul, les modèles de communication et de synchronisation, la modularité, l'expression du parallélisme, la notion de temps …

♦ Le deuxième problème concerne le « gap » et la « discontinuité de modèles » qui peut exister, entre la spécification système et le niveau implémentation. La spécification système doit faire abstraction des détails d'implémentation afin de retarder autant que possible les choix concernant la réalisation pour n'exclure aucune alternative possible. Une implantation peut être vue comme une transcription d'une spécification de haut niveau vers une description bas niveau. Plusieurs étapes de raffinement sont alors nécessaires pour obtenir une implémentation efficace. A chaque étape, on choisit une réalisation d'un ensemble de détails réduisant ainsi le fossé entre spécification et réalisation. En effet, les concepts au niveau système, sont généraux et une transcription naïve (directe) de ces concepts aboutit inévitablement à une solution irréalisable.

♦ Le troisième problème est lié directement au deuxième problème et concerne l'automatisation des étapes de raffinement. Le passage manuel de la spécification à l'implémentation est ardu et requiert des compétences multidisciplinaires. Il peut-être source d'erreurs complexes difficilement détectable (composition de l'application parallèle, logiciel système et le matériel multiprocesseur). Ce passage n'est plus applicable à partir d'un certain degré de complexité. La conception d'architecture consiste surtout à assembler automatiquement des composants existants. Ainsi il devient crucial de combiner la conception des couches logicielles et matérielles (interface matériel/logiciel, logiciel application, logiciel système, et composants de calcul et de mémorisation).

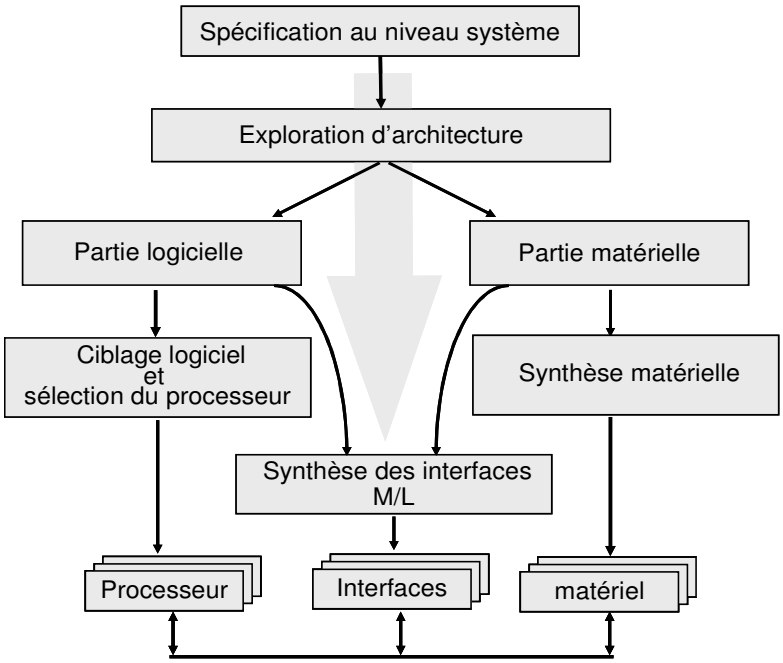


Figure 1. 2: Flot de conception de haut niveau des MPSoC

Face à ces difficultés, il faut proposer des méthodologies, trouver des solutions techniques et développer des outils pour transposer la spécification vers une implémentation sur une architecture MPSoC. La Figure 1. 2 présente un flot générique de conception de haut niveau des MPSoC. Le flot part d'une spécification fonctionnelle de l'application qui permet de fixer les principales contraintes du produit à réaliser. Ce modèle sera aussi utiliser pour une exploration des algorithmes et fixer certains paramètres architecturaux. L'exploration de l'architecture consiste à trouver le meilleur partitionnement logiciel/matériel, l'allocation des processus et le choix des protocoles et des réseaux de communication. L'application est ainsi raffinée à un niveau intermédiaire qui contient les paramètres de l'architecture choisie. A partir de ce niveau, l'implémentation permet de générer l'architecture RTL. Ceci comporte la synthèse de la partie matérielle, la compilation de la partie logicielle pour cibler un processeur bien défini, et le raffinement de la communication qui génère des interfaces matérielles/logicielles connectant les différents composants au réseau de communication. En parallèle, il est observé que plus la validation intervient tôt dans le flot de conception d'un système, plus une erreur peut être corrigée rapidement. Afin de réduire les coûts de

conception et d'économiser les efforts inutiles, il est important de vérifier la fonctionnalité du système à tous les niveaux durant le processus de conception.

3. Objectifs et contributions

La Figure 1.3 illustre le « gap » entre le modèle au niveau système et les modèles d'architectures et la « discontinuité » entre les outils de conception et d'exploration des algorithmes (Simulink/Matlab dans notre cas) et les outils de conception des architectures (Roses et Macrocell Builder dans notre cas). Le gap est dû au fait que la spécification système exécutable décrit le comportement (fonctionnalités) des algorithmes, en manipulant des concepts qui sont généraux. Une transcription naïve (directe) de ces concepts aboutit inévitablement à une solution irréalisable. La discontinuité est due au fait que les modèles au niveau systèmes utilisés par les outils d'exploration d'algorithmes ne sont pas forcement en adéquation avec les modèles d'architecture utilisés par les outils de conception d'architecture. Il se trouve même que les modèles utilisés aux deux niveaux d'abstraction sont complètement décorrélées.

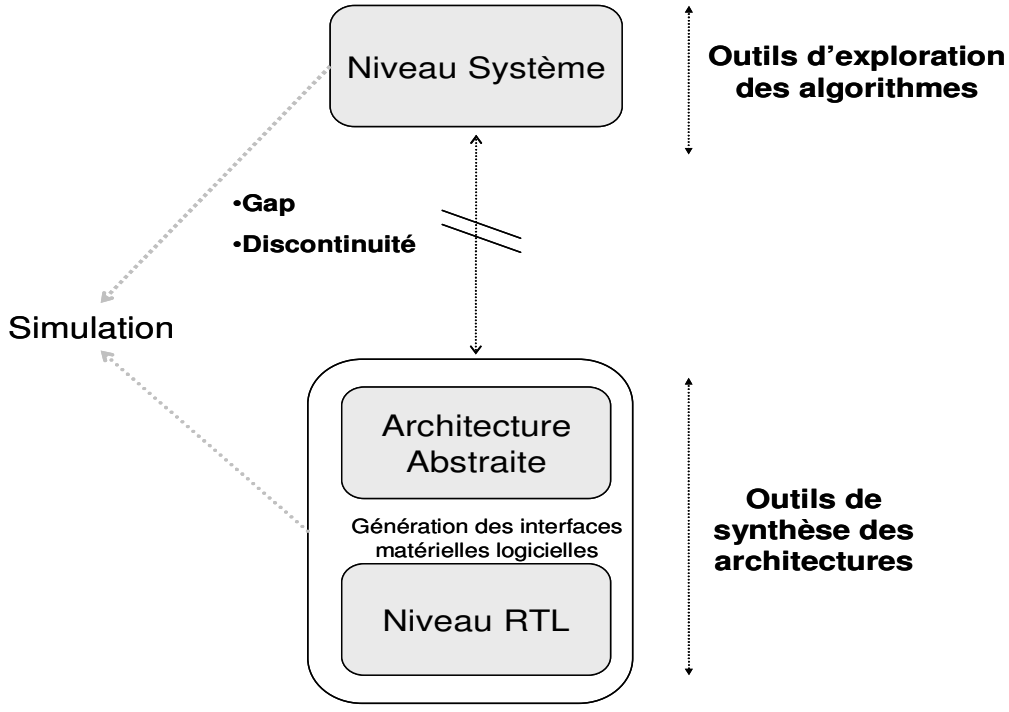


Figure 1.3: Gap et discontinuité entre le niveau système et le niveau architecture

L'objectif global de cette thèse est la mise en œuvre d'une méthodologie et d'un flot de conception de haut niveau des MPSoC à partir de Simulink. Plus précisément, il s'agit d'élaborer des approches et des stratégies de conception permettant de résoudre les problèmes cités précédemment : réduire le gap entre le modèle au niveau système et les modèles d'architectures, et résoudre le problème de la discontinuité entre les outils de conception et d'exploration des algorithmes (Simulink/Matlab) et les outils de conception des architectures (Roses et Macrocell Builder).

Trois contributions ont été définies à travers cette thèse pour atteindre cet objectif :

1) La première contribution concerne les « modèles Simulink » et peut-être séparée en deux parties distinctes : le « modèle fonctionnel » et le « modèle au niveau transactionnel ». Pour la modélisation fonctionnelle en Simulink, nous définissons un sous-ensemble de Matlab/Simulink et associons un ensemble de règles de description (concernant le niveau de parallélisme, l'abstraction des données, le pipeline, le contrôle) permettant la spécification et la validation fonctionnelle efficace des algorithmes de l'application. Pour réduire le « gap » entre le modèle fonctionnel et le modèle d'architecture en SystemC, nous proposons un nouveau model intermédiaire transactionnel exécutable dans l'environnement Simulink qui combine à la fois l'algorithme et l'architecture dans un même modèle de représentation. Il permet d'une part l'exploration rapide d'architecture dans l'environnement Simulink et d'autre part de préserver la « continuité » entre le niveau système et le niveau architecture. Il permet ainsi d'établir un lien entre les outils de conception des algorithmes et les outils de conception des architectures.

2) La deuxième contribution est la génération automatique de la macro architecture à partir du modèle mixte algorithme/architecture transactionnel en Simulink. Cette macro architecture est le point d'entrée des outils de conception d'architecture : Roses pour les interfaces matériel/logiciel et MacroCell Builder pour la conception du matériel dédié. Cet outil permet ainsi de grouper tous les outils dans un flot complet qui part de la spécification fonctionnelle de l'algorithme jusqu'à l'intégration du système complet matériel/logiciel. Il garanti, ainsi, une continuité dans la méthodologie de conception de haut niveau.

3) La dernière contribution de cette thèse est la mise en œuvre d'une application en vraie grandeur, « codec standard MPEG Layer III » : La spécification fonctionnelle parallèle et « pipeliné » sous Matlab/Simulink ; le découpage des algorithmes en blocs réutilisables et paramétrables intégrant des « *S-functions* » ; la description du modèle intermédiaire TLM et l'exploration algorithme/architecture ; la génération de macro architecture, la génération des interfaces matériel/logiciel ; la génération de micro architecture ; la simulation et la validation à tous les niveaux d'abstractions jusqu'au niveau RTL. Cette application de vraie grandeur a servi notamment à expérimenter la méthodologie et le flot de conception. Elle a permis de valider l'interopérabilité entre plusieurs outils, de détecter et de corriger plusieurs bugs.

4. Plan et organisation

Ce manuscrit est organisé comme suit :

Dans le chapitre 2, nous présentons les méthodologies et les flots de conception existants spécifiques à notre domaine et nous situons notre contribution par rapport à cet état de l'art. Nous commençons par un rappel de quelques définitions importantes sur les caractéristiques et les spécificités des systèmes logiciel/matériel embarqués sur puce. Nous discutons, ensuite, des caractéristiques requises des méthodologies de conception des MPSoC et nous présentons les principales étapes de conception. Enfin, nous analysons quelques outils et méthodologies académiques et industriels que nous considérons les plus représentatifs du domaine.

Dans le chapitre 3, nous présentons la méthodologie et le flot de conception proposés. Le modèle d'entrée est une description fonctionnelle en Simulink/Matlab. Le modèle de sortie est un modèle virtuel au niveau cycle CA (Cycle Accurate) du système matériel/logiciel (c'est-à-dire le matériel dédié et les interfaces matérielles sont au niveau transfert de registres RTL, le logiciel est exécuté sur des simulateurs de processeurs ISS au niveau CA). Quatre niveaux d'abstraction sont définis, permettant les raffinements et les validations progressifs à travers le flot de conception. Enfin, nous illustrons les principaux outils qui forment notre flot (Simulink/Matlab pour la spécification fonctionnelle et le modèle transactionnel intermédiaire algorithme/architecture, Roses

pour la conception des interfaces matériel/logiciel, Macrocell Builder pour conception du matériel dédié).

Dans le chapitre 4, nous présentons l'outil de génération automatique des macro architectures. Cet outil permet d'établir le lien entre « l'environnement Simulink » et les outils « Roses et Macrocell Builder ». L'entrée de cet outil est un modèle mixte algorithme/architecture transactionnel en Simulink. Ce modèle, qui sera détaillé dans ce chapitre, est décrit sous forme de blocs interconnectés entre eux via des vecteurs de données et regroupés dans des sous systèmes. La génération de la macro architecture s'effectue en plusieurs étapes grâce à l'association de trois composantes indépendantes de l'outil de génération de la macro architecture. Nous détaillons les différentes phases de génération et les composantes associées.

Le chapitre 5 est consacré à l'expérimentation de la méthodologie et le flot de conception sur une application en vraie grandeur «décodeur MPEG Layer III ». Nous présentons et retraçons toutes étapes importantes qui nous ont conduit à la validation de l'approche sur cette application, avant l'étape finale de l'intégration du système MPSoC matériel/logiciel.

Enfin, nous consacrons le dernier chapitre aux conclusions et aux perspectives de cette thèse.

CHAPITRE 2

LES ARCHITECTURES MULTIPROCESSEUR MONOPUCE : DE LA SPECIFICATION A L'IMPLEMENTATION

Sommaire

1. Introduction

Les applications embarquées appartiennent à un domaine en très forte expansion. Néanmoins elles sont de plus en plus soumises à des fortes contraintes fonctionnelles (Puissance de calcul, consommation, miniaturisation,..) et non fonctionnelles (tels que, temps de mise sur le marché, forte croissance de la quantité de la production). D'un autre coté, le progrès technologique permettra une capacité d'intégration de centaines de millions de transistors sur une seule puce. Ce progrès technologique détermine ce qui est possible; et c'est l'architecture associée à un flot de conception efficace qui traduit le potentiel de la technologie en performance et capacité. Concernant l'architecture, les trois manières dont un grand volume de ressources améliore la performance sont le parallélisme, la localité, et la spécialisation. Le parallélisme implique multiprocesseur, la localité implique monopuce, et la spécialisation implique dédiée à une application spécifique. Quant au flot de conception, manipuler un grand volume de ressources revient à remonter le niveau d'abstraction et proposer une méthodologie systématique pour le passage de ce niveau à une implémentation optimale.

Ce chapitre présente les systèmes multiprocesseurs monopuces en indiquant leurs propriétés. Ensuite, il expose les principales étapes de conception, pour dévoiler par la suite les diverses caractéristiques requises par les outils de conception des SoCs. Enfin, une étude des outils de conception existants sera présentée pour donner une idée sur quelques travaux récents académiques et industriels.

2. Systèmes monopuces

Un système monopuce, appelé encore SoC (System-on-Chip) ou système sur puce, désigne l'intégration d'un système complet sur une seule pièce de silicium. Ce terme est devenu très populaire dans le milieu industriel malgré l'absence d'une définition technique standard [Ron 99]. Dans le dictionnaire le mot système fait référence à un assemblage d'éléments qui se coordonnent pour concourir à un résultat. En grec ´ sustêma' signifie ensemble. Ce mot provient du verbe ´ synistanai ' qui veut dire combiner, établir, rassembler [Bou 06]. Une définition plus appropriée de système monopuce serait : Un système complet sur une seule pièce de silicium, résultant de la

cohabitation sur silicium de nombreuses fonctions déjà complexes en elles mêmes : processeurs, DSP, mémoires, bus, convertisseurs, blocs analogiques, etc. Il doit comporter au minimum une unité de traitement de logiciel (un CPU) et doit dépendre d'aucun (ou de très peu) de composants externes pour exécuter sa tâche.

La Figure 2.1 présente un exemple de système monopuce typique. Il se compose d'un cœur de processeur (CPU), d'un processeur de signal numérique (DSP), de la mémoire embarquée, et de quelques périphériques tels qu'un DMA et un contrôleur d'E/S. Le CPU peut exécuter plusieurs tâches via l'intégration d'un OS. Le DSP est habituellement responsable de décharger le CPU en faisant le calcul numérique sur les signaux de provenance du convertisseur A/N.

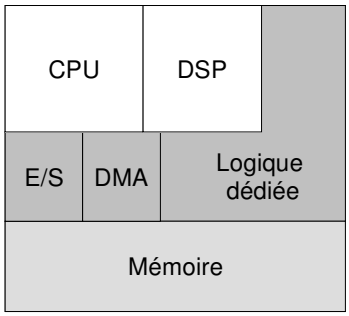


Figure 2.1 Un exemple de système monopuce (SoC)

Le système pourrait être construit exclusivement de composants existants, ou d'une combinaison de composants existants et de solutions faites sur mesure.

2.1 Caractéristiques des SOCs

2.1.1 Hétérogénéité des composants du SoC

Le système monopuce n'encapsule pas des systèmes purement électroniques, il peut intégrer aussi des composants non électroniques. C'est le cas, par exemple, des systèmes intégrant des éléments micro-mécaniques (MEMS de l'anglais Micro-Electro-Mechanical Systems). La portion électronique dominante dans un SoC peut être formée d'une partie analogique et d'une partie numérique.

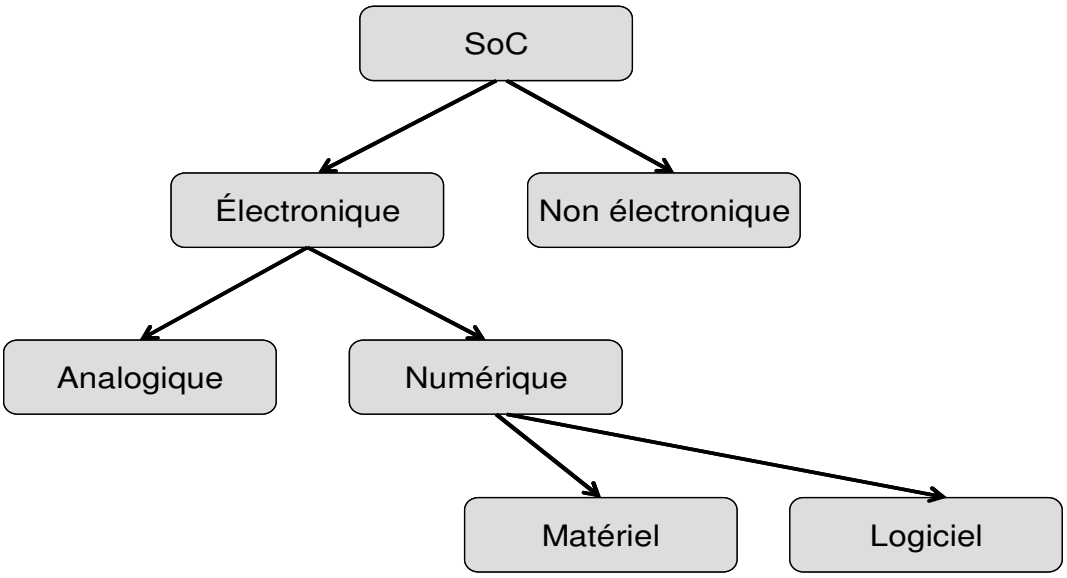


Figure 2.2 : Les composants hétérogènes formants un SoC

Le téléphone portable présente un exemple quotidien de tels systèmes hybrides où la partie radio (analogique) et la partie de traitement de signal de l'interface utilisateur (numérique) cohabitent sur la même puce. La partie numérique, peut être décomposée selon qu'il s'agit d'un traitement effectué par un programme logiciel tournant sur un (ou plusieurs) processeur(s) représentant ainsi la partie logicielle, ou d'une fonction directement ciblée en matériel. Ces divers choix sont illustrés dans la Figure 2.2.

2.1.2 L'embarquement du SoC dans un système complet

Un système embarqué est un sous-système enfoui dans un système plus large pour procurer des fonctions particulières. L'interaction avec le reste du système se fait à travers des interfaces d'adaptation. Les systèmes monopuces sont des systèmes embarqués (la réciproque ne tient pas forcément) dans le sens où ils sont toujours appelés à évoluer dans le contexte d'un système plus large qui constitue l'environnement du système monopuce. L'interaction entre le SoC (numérique) est les éléments de l'environnement (capteurs et actionneurs) se fait généralement via des convertisseurs analogique/numérique.

2.1.3 La spécialisation des SoCs suivant les applications implémentées

Un système monopuce est par définition même, différent des systèmes à usages généraux tel que les ordinateurs. Par opposition à ces systèmes d'usage général, un SoC cible toujours une application particulière bien définie. On ne cherche donc pas qu'un SoC soit polyvalent mais plutôt d'être spécifique. Cette distinction implique des différences radicales au niveau des méthodologies de conception entre un système monopuce est un système à usage général [Jer 04']. Notons cependant qu'une telle séparation, si elle est franche dans une grande partie d'applications, l'est beaucoup moins dans certains autres types d'application qui héritent à la fois des caractéristiques des deux univers, tels que les assistants personnels (PDA), etc.

Il est important également de souligner que même si le caractère spécifique des SoCs signifie qu'ils sont taillés sur mesure, cela n'implique pas forcément qu'ils sont conçus à chaque fois en partant de zéro. En effet, les méthodologies de conception des SoCs font souvent appel à la réutilisation qui ne contredit pas le caractère spécifique des systèmes en question.

2.1.4 Réduction de la latence

Il y a plusieurs raisons pour lesquelles la solution monopuce représente une manière attrayante pour implémenter un système. Les processus de fabrication d'aujourd'hui (de plus en plus fins) permettent de combiner la logique et la mémoire sur une seule puce, réduisant ainsi le temps global des accès mémoire. Étant donné que le besoin en mémoire de l'application ne dépasse pas la taille de la mémoire embarquée sur la puce, la latence de mémoire sera réduite grâce à l'élimination du trafic de données entre des puces séparées.

2.1.5 Réduction du coût

Puisqu'il n'y a plus aucun besoin d'accéder à la mémoire sur des puces externes, le nombre de broches peut également être réduit et l'utilisation de bus sur carte devient obsolète. Le coût de l'encapsulation représente environ 50% du coût global du processus de fabrication de puce [Rea']. Par rapport à un système- sur- carte ordinaire, un SoC emploie une seule puce réduisant le coût total d'encapsulation, et de ce fait, le coût total

de fabrication. Ces caractéristiques aussi bien que la faible consommation et la courte durée de conception permettent une mise sur le marché rapide de produits plus économiques et plus performants.

2.2 Du monoprocesseur au Multiprocesseur

Avec le progrès technologique et la capacité d'intégration de centaines de millions de transistors sur une seule puce deux tendances architecturales ont émergé pour relever ce défi. La première tendance s'est restreinte à l'utilisation d'architectures monoprocesseurs tout en améliorant considérablement les performances du CPU utilisé et l'utilisation de coprocesseurs. Un tel CPU se distingue par : une fréquence de fonctionnement très élevée, des structures matérielles spécialisées, un ensemble d'instructions sophistiquées, plusieurs niveaux de hiérarchie mémoire (plusieurs niveaux de caches), et des techniques spécialisées d'optimisation logicielle (nombre d'accès mémoire, taille, etc.). Dans cette direction on peut citer : PowerPC, Intel Pentium 4, ST100, et beaucoup d'autres processeurs de type VLIW ou superscalaire. La Figure 2.3 présente un exemple d'architecture conventionnelle µP/co-processeurs. Dans de telles architectures, la communication est basée sur le principe maître/esclaves : le CPU est le maître et les périphériques sont les esclaves. Les interfaces des co-processeurs sont généralement faites de registres transposés dans la mémoire du CPU et peuvent produire des interruptions au CPU. Ces communications se font généralement via un bus partagé (le bus mémoire du CPU). En termes de performance, de telles architectures –centrées autour d'un seul CPU– ont un inconvénient : la dégradation de performance engendrée par le fait que le processeur effectue la communication aussi bien que le calcul.

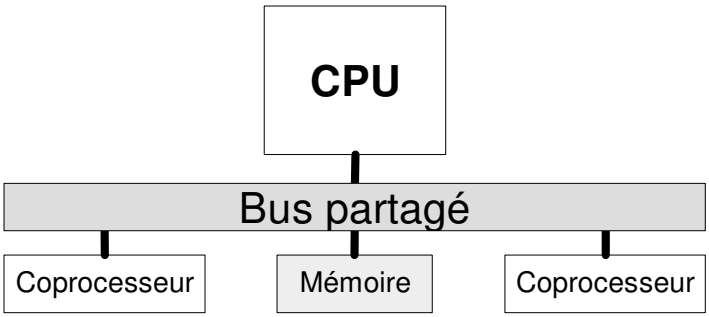


Figure 2.3 : Un exemple d'architecture monoprocesseur (µP/co-processeurs)

La deuxième tendance s'est tournée vers les architectures multiprocesseurs (multimaîtres). Ces architectures qui étaient réservées jusqu'à maintenant aux machines de calculs scientifiques (tel que CM [Hil 93], CRAY [Sco 96], etc.) ne le sont plus avec le progrès de la technologie. Ainsi, l'idée de ceux qui ont choisi d'adopter cette tendance est la suivante : En regardant plus étroitement les avancements en technologie et architectures sous-jacentes, il s'avère qu'il peut être de plus en plus difficile d'avoir des architectures monoprocesseurs assez rapides alors que les architectures multiprocesseurs sur puce deviennent déjà réalisables.

2.3 Domaines d'application des SoCs

Le domaine d'application couvert par des systèmes multiprocesseurs sur puce est très vaste, puisqu'il comprend toutes les applications dont les impératifs économiques et les contraintes de conception nécessitent une architecture mixte, miniature et efficace. Les applications télécommunications et multimédias sont les plus représentatives de ce domaine. Elles constituent l'un des marchés le plus en expansion actuellement (modems, téléphones mobiles, les décodeurs audio et vidéo, etc.), notamment par le développement des télécommunications sans fil et d'Internet. Il constitue l'une des plus intéressantes applications, car il nécessite la production au moindre coût de systèmes à contraintes strictes (performance, poids, consommation, etc.) et ceci doit être fait dans un environnement de grande concurrence ou un temps de mise sur le marché court est absolument crucial. Les produits sont nécessairement fournis en de nombreuses versions différentes mises à jour continuellement. Il est par conséquent important que les composants existants puissent être réutilisés aussi souvent que possible, d'où le besoin en produits flexibles conçus à faible coût.

3. Principales étapes de la conception des MPSoCs

Il s'agit essentiellement de la spécification, la modélisation, le partitionnement, le raffinement du logiciel, le raffinement du matériel, la génération d'interface entre le logiciel et le matériel (i.e raffinement des communications) et la génération de code ou le prototypage. La succession de ces étapes forme le flot typique d'une approche de conception des systèmes multiprocesseurs schématisée dans la Figure 2.4.

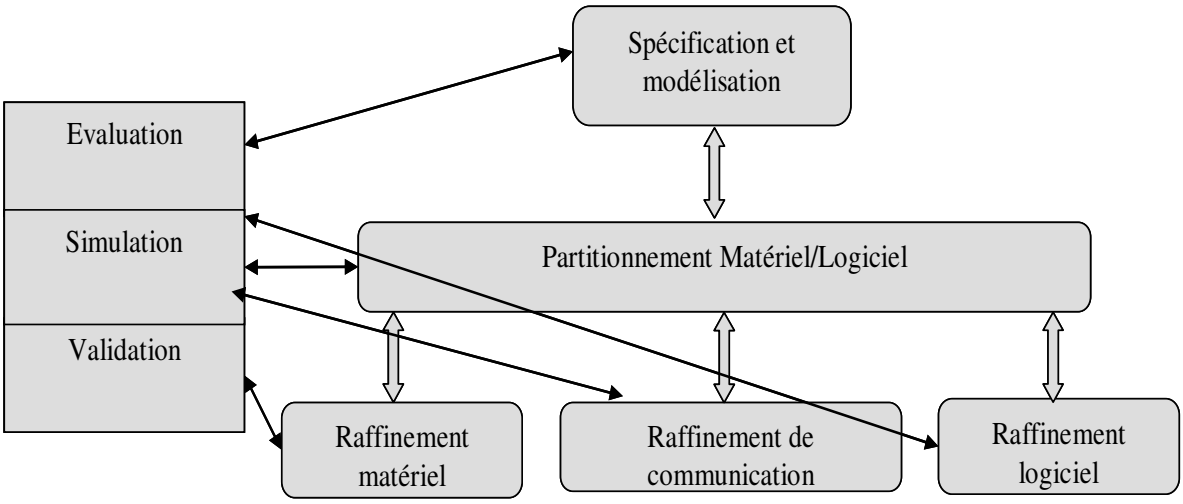


Figure 2.4 : Flot de conception des SoCs

Généralement la conception des systèmes sur puce part d'une spécification unique décrivant l'architecture et/ou le comportement du système à concevoir [Jer 04']. Après cette phase de spécification survient l'étape de partitionnement du système ayant pour but de décomposer ce dernier en trois parties :

- Une partie matérielle implémentée sous forme de circuits (FPGA, ASIC, ...).
- Une partie logicielle implémentée sous forme d'un programme exécutable sur processeur, par exemple à usage général.
- Une interface de communication entre les parties matérielles et logicielles.

Les trois parties obtenues doivent ensuite être vérifiées et validées avant de passer à la phase du raffinement et d'implémentation. Il est nécessaire de faire des retours aux étapes précédentes (feed-back), plus précisément à l'étape de partitionnement, tant que l'architecture obtenue ne répond pas aux contraintes auparavant fixées.

3.1 Spécification et modélisation

La spécification est le point de départ du processus de conception des systèmes sur puce. Cette étape consiste, en général, à décrire les fonctionnalités du système à concevoir ainsi que toutes les contraintes qu'il doit satisfaire sans se soucier du découpage matériel/logiciel qui en suit. Le résultat de cette étape est une spécification fonctionnelle du système indépendante de tout détail d'implémentation future, qu'elle soit matérielle ou logicielle. Cette spécification unifiée constitue donc le premier pas vers

l'unification de la conception dans un processus de co-design. Il est donc indispensable que le langage de description utilisé ainsi que le modèle interne sous-jacent, adopté pour représenter le système soient assez puissants pour inclure le logiciel, le matériel, ainsi que toutes les contraintes associées au système. En fonction du modèle interne adopté pour la représentation du système spécifié, on distingue dans la littérature différentes approches de conception des SoCs [Jer 04'], [Cos 00]. La première classe correspond aux environnements de conception de SoCs basés sur un modèle de spécification interne homogène (homogeneous approach), le système est décrit par une spécification unique et unifiée indépendamment de toute considération matérielle ou logicielle. La seconde classe correspond aux environnements de conception conjointe basés sur l'utilisation de modèles multiples pour spécifier les différentes parties du système. Chaque partie du système est spécifiée à l'aide d'un modèle correspondant à sa nature (heterogeneous approach). Il est important de noter qu'une autre classification possible est basée sur le nombre de langages de spécification utilisés [Moo 97]. On distingue ainsi les approches homogènes (basées sur l'utilisation d'un langage de spécification unique) des approches hétérogènes (basées sur l'utilisation de langages spécifiques à chaque partie du système). Cependant, nous considérons qu'une telle classification est un peu trop superficielle, étant donnée que souvent plusieurs langages de spécification utilisés au début du processus de spécification peuvent être basés sur le même modèle de représentation interne. Enfin, il est intéressant de noter que le choix du formalisme (langage et modèle) de spécification utilisé est d'une importance primordiale, car il peut avoir une influence directe sur les autres étapes de conception, plus particulièrement le partitionnement. Ce choix du formalisme de spécification constitue aussi un premier pas vers l'unification de la conception.

3.2 Implémentation

Nous appelons l'implémentation la réalisation physique du matériel (par la synthèse) et du logiciel exécutable (par la compilation). Une caractéristique désirable des approches de co-design lors de l'implémentation est qu'il est préférable qu'il n'y ait pas un problème de continuité de modèle afin de préserver les propriétés formelles initiales ; c.à.d, les étapes successives de raffinement de la modélisation à la synthèse devraient toutes

s'articuler autour du même modèle interne adopté en fixant à chaque étape certains détails d'implémentation [Sch 00]. Les informations architecturales concernant les détails d'implémentation seront prises en compte par le modèle lors de la phase d'implémentation.

Les étapes de cette phase d'implémentation sont principalement :

- Le partitionnement matériel/logiciel

-Le raffinement et la synthèse du code pour la partie matérielle et la partie logicielle

3.3 Partitionnement logiciel/matériel

Après la phase de spécification survient une phase de partitionnement ayant pour but d'assurer la transformation des spécifications du système en une architecture composée d'une partie matérielle et d'une partie logicielle. Cette transformation s'effectue habituellement en deux phases : la sélection d'une architecture multi composants (la plateforme) constituée souvent de composants matériels (ASICs, FPGAs, composants standards...) et de composants logiciels (ASIP, DSP...), et l'allocation des éléments du modèle fonctionnel spécifiant l'application sur les composants de cette architecture. Cette phase de partitionnement consiste donc à déterminer les parties du système qui seront réalisées en matériel et celles qui seront réalisées en logiciel ainsi que l'interface entre ces différentes parties.

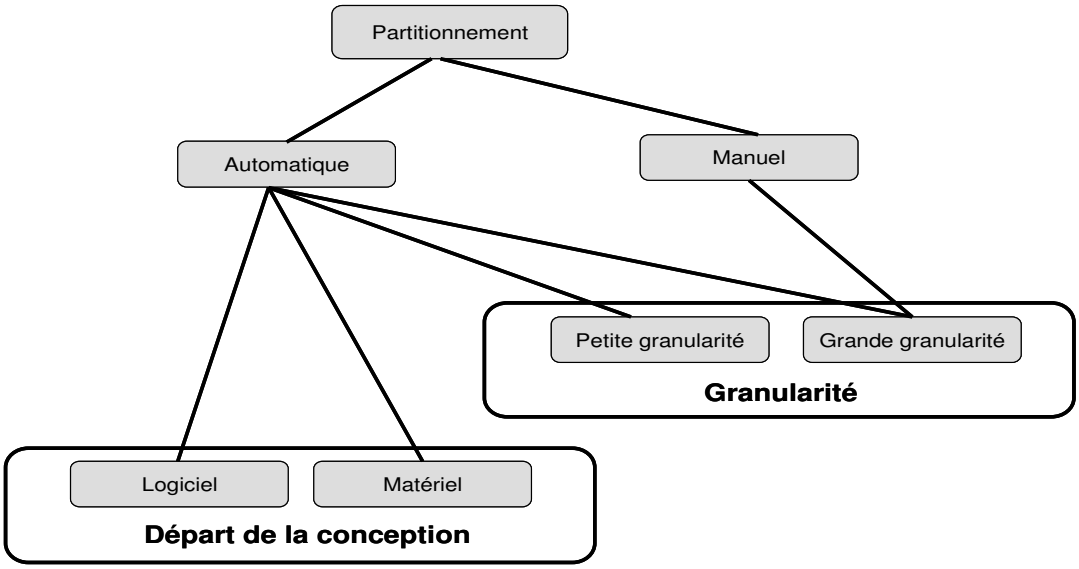


Figure 2.5:Approche de partitionnement

La définition de ces différentes partitions est souvent guidée par l'ensemble des contraintes définies initialement par le concepteur (espace réservé au logiciel et au matériel, temps d'exécution, taux de parallélisme, taux de communication, consommation, temps de conception et coût industriel, poids,...). En effet, une fonctionnalité donnée peut avoir diverses implémentations (en matériel ou en logiciel), chacune avec ses propres caractéristiques de coût et performance. Il est donc nécessaire d'explorer différentes possibilités d'implémentation en matériel et en logiciel avant de se décider sur la partition du système qui satisfait de façon optimale les contraintes de coût et/ou de performance imposées. Le partitionnement est donc d'une importance critique car il a un impact important sur les caractéristiques de coût/performances du système final. Les nombreuses méthodes adoptées dans la littérature pour résoudre cette problématique de partitionnement logiciel/matériel varient toutes sur l'approche utilisée. Nous pouvons tout de même les classer comme il est montré sur la Figure 2.5 en fonction d'un certain nombre de facteurs tels que le degré de granularité, le fait que le partitionnement soit manuel ou automatique.

Le degré de granularité indique en fait le niveau de finesse des éléments du modèle fonctionnel adopté lors du processus de partitionnement. Ce dernier varie souvent du grain fin "fine -grained" (niveau opérations, instructions), au grain moyen "medium-grained" (niveau macro- opérations, blocs de base...), au gros grain "coarse-grained" (niveau processus, tâches, fonctions,...). Il n'existe cependant pas de consensus quant au niveau à adopter pour le partitionnement, car chacun présente des avantages et des inconvénients. Toutefois, les approches interactives adoptent souvent une granularité de haut niveau afin de pouvoir traiter de manière efficace des systèmes de grandes tailles pour lesquels il est de plus en plus difficile de gérer manuellement un partitionnement à grain fin. Les approches automatiques basées sur des heuristiques guidées par leurs fonctions de coût quant à elles adoptent souvent une granularité plus fine.

3.4 Raffinement

Une fois le partitionnement logiciel/matériel terminé les décisions liées à l'implémentation souhaitée sont fixées, l'étape de raffinement se chargera de transformer les spécifications fonctionnelles en descriptions directement implantables sur les

composants matériels et logiciels de l'architecture cible. Son rôle est d'assurer la conception physique de la partie matérielle (par synthèse de haut niveau), la génération du code exécutable correspondant à l'implémentation de la partie logicielle (par la compilation) ainsi que le raffinement des interfaces matériel/logiciel en vue de produire finalement selon le cas un prototype physique du système conçu ou le produit fini lui même. Pour ce faire, le processus de raffinement doit pouvoir s'interfacer et se connecter aux outils de développement logiciel et matériel existants supportés par les composants constituant l'architecture d'exécution du système conçu. Cet interfaçage est rendu possible via la mise en œuvre d'une génération automatique de code source accepté comme entrée par ces outils.

3.4.1 Raffinement du logiciel

Le raffinement logiciel consiste en la synthèse et la génération d'un code exécutable correct et efficace correspondant à l'implémentation de la partie logicielle à partir d'une spécification de haut niveau. Les intérêts de ce raffinement logiciel résident dans la diminution du coût de développement et surtout l'augmentation de la fiabilité du code généré. La complexité de cette étape dépend du type de processeur utilisé. On distingue souvent la synthèse du logiciel pour les processeurs généraux (Pentium, SPARC,...), de la synthèse pour les processeurs de traitement du signal et de la synthèse pour les ASIP (Application-Specifc Instruction set- Processsors).

Le plus souvent, une méthode efficace de synthèse consiste à définir un schéma d'implémentation logicielle sans utiliser d'exécutifs temps- réel. Une autre méthode consiste à utiliser un exécutif temps -réel. Une solution intermédiaire existe qui utilise judicieusement les qualités de ces deux méthodes.

3.4.2 Raffinement du matériel

Le raffinement matériel consiste à transformer la spécification de haut niveau de l'application vers un circuit électrique. Cette opération est appelée la synthèse du matériel. On distingue généralement deux niveaux de synthèse matérielle : synthèse logique et synthèse comportementale. La synthèse logique consiste en la transformation d'une description de niveau RTL (Register Transfer Level) en un réseau de portes

logiques interconnectées qui réalise les fonctionnalités souhaitées. La synthèse comportementale consiste quant à elle en la transformation d'une description comportementale faite en langage de haut niveau (un algorithme) vers une architecture décrite au niveau RTL, composée d'une partie chemin de données et d'une partie chemin de contrôle.

Du fait que lors de la conception le raffinement du matériel et du logiciel sont traitées globalement de façon identique et à des niveaux d'abstraction similaires, le raffinement du matériel dans le cadre de la conception des SoCs repose sur la synthèse au niveau comportemental (i.e synthèse d'architecture au niveau RTL). Les outils classiques de CAO permettent par la suite le passage quasi automatique du modèle RTL ainsi généré à la puce réalisant ainsi la synthèse logique. Les intérêts de cette synthèse automatique du matériel résident dans l'augmentation de la productivité du matériel, la diminution du coût de développement et surtout l'augmentation de la fiabilité du circuit généré.

3.4.3 Raffinement d'interface ou raffinement des communications

Le raffinement des communications est la réalisation des interfaces de communications entre les différentes ressources. Plus précisément, elle permet de définir les protocoles de communications et les interfaces d'E/S entre les différentes partitions (sous-systèmes communicants issus du partitionnement). En effet, les différentes parties du système échangent des informations à travers des mécanismes dédiés à la communication. Ces mécanismes peuvent être mis en ouvres en matériel (bus, contrôleurs...) ou en logiciel (pilotes, protocoles,...) [Gra 04]. Certains concepteurs préconisent l'utilisation de protocoles "standard" ou tout au moins connus, alors que d'autres développent leurs propres protocoles.

3.5 Vérification et Validation

A ce stade, l'activité de validation permet de vérifier que les sous-systèmes obtenus après les étapes successives de partitionnement et de synthèse réalisent lorsqu'ils fonctionnent ensemble les comportements attendus du système global et que l'implémentation obtenue préserve bien les propriétés du système spécifié. Signalons tout de même qu'une telle activité de validation est transversale à tout le processus de

conception des SoCs. A La fin de chaque étape du processus, il est nécessaire d'évaluer si les objectifs de conception ont été atteints pour pouvoir procéder sans risque à la prochaine étape. Cette validation permet donc d'évaluer si les critères nécessaires pour passer à la prochaine étape sont satisfaits ou pas. Si ce n'est pas le cas, le concepteur doit réitérer l'étape actuelle jusqu'à ce qu'il obtienne une validation réussie. Les activités de validation à la fin de chaque étape bien qu'elles ont toutes les mêmes objectifs (i.e vérifier si la conception est acceptable), elles diffèrent néanmoins sur le type d'approche, les outils etc. Le fait d'avoir une activité de validation à la fin de chaque étape assure l'homogénéité de la conception vis à vis du respect des objectifs de l'ensemble des étapes précédentes de conception. L'activité de validation peut être effectuée selon trois approches différentes :

- La vérification formelle par l'application des méthodes de preuves formelles à différents niveaux d'abstraction permettant de vérifier si la spécification est complète et correcte et de vérifier la conformité entre la spécification et l'implémentation finale.
- La simulation par l'intermédiaire d'outil de co-simulation permettant la simulation conjointe de la partie logicielle, de la partie matérielle et de l'interface entre les deux.
- Le prototypage par la génération rapide d'une implémentation prototype permettant d'évaluer, de valider la faisabilité du système ainsi que ses caractéristiques.

Pour conclure cette description des différentes phases du processus de conception des SoCs, il est intéressant de noter que contrairement aux premières approches, dans lesquelles la recherche en co-design avait pour objectif la transformation automatique d'une spécification en une implémentation, les chercheurs tendent actuellement à mettre l'accent sur des problèmes plus spécifiques liées au processus de conception, en particulier, la spécification, la modélisation architecturale, le partitionnement, l'estimation des caractéristiques, les heuristiques d'exploration, la synthèse de la communication, la validation, et la génération automatique de code.

4. Caractéristiques requises des outils de conception des SoCs

Il est indispensable de rappeler ici les principales exigences et caractéristiques que doit posséder un système de conception conjointe. Une telle liste de caractéristiques pourrait

servir à indiquer les défis actuels ainsi que les domaines qui posent toujours des problématiques et méritent une attention particulière des efforts des futurs travaux de recherche. Ces caractéristiques portent sur plusieurs aspects du processus de la conception conjointe.

4.1 Approche du modèle intermédiaire

Pour s'affranchir de la diversité et de l'évolution constante des langages de spécification, il est de plus en plus souhaitable, voir indispensable, d'adopter une approche de conception basée sur l'utilisation d'un modèle de représentation interne autour duquel s'articuleront les différentes activités du processus de conception des SoCs. Ainsi, tous les traitements des différentes étapes du processus de raffinement de la spécification à l'implémentation, seront effectués sur le modèle interne décrivant le système à concevoir. Une telle approche indépendante des langages de spécification s'avère être de plus en plus une solution attractive pour la conception des systèmes sur puce. La Figure 2.6 illustre la coordination de plusieurs langages pour aboutir à un langage intermédiaire.

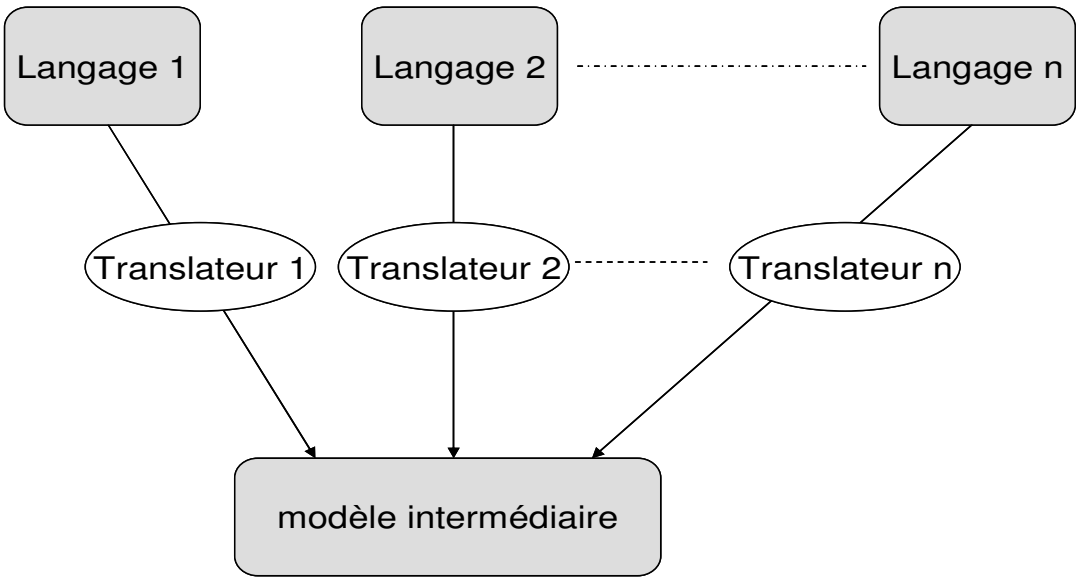


Figure 2.6: La diversité des langages et le modèle intermédiaire

4.1.1 Différence entre un langage de spécification et un modèle interne

Certains concepteurs confondent le langage de spécification et le modèle interne et d'autre ne voit donc pas la nécessité d'avoir un modèle interne à leur environnement de conception. Dans une telle situation, toutes les opérations sur le code décrivant le système à concevoir sont effectués dans le langage de spécification choisi [Dia 97], [Dem 97], ce qui limite considérablement les environnements de conception conjointe qu'ils proposent. Edwards [Edw 97] affirme que la distinction entre le langage et le modèle de calcul sous-jacent est important et Stoy [Sto 95] considère que le fait que des approches de co-design n'incluant pas de modèle de représentation défini formellement signifie qu'elles sont construites sur des bases très instables. La définition et le choix judicieux d'un modèle de représentation interne pour un système de conception de SoC est donc d'une importance primordiale, car il peut avoir une influence directe sur le déroulement des différentes étapes du processus de conception et par conséquent sur les performances et la qualité du système conçu.

4.1.2 Choix du Modèle interne

Ce choix est souvent guidé par plusieurs considérations :

-La continuité du modèle à travers les étapes de conception

Tout d'abord, un processus de conception est souvent perçu comme une séquence de tâches de raffinement caractérisées par une transformation d'une spécification plus abstraite en une spécification plus détaillée. Il est donc désirable de disposer d'un modèle de spécification permettant de passer aisément d'une spécification à une autre sans avoir à apprendre de nouveaux concepts. Ceci n'est possible qu'à travers l'adoption d'un même modèle de représentation interne durant la totalité du processus de conception dont la sémantique évolue au fur et à mesure du raffinement (ce qui correspond au concept de "continuité du modèle"). Un tel modèle doit être capable de fournir les concepts de tous les niveaux d'abstraction et de capturer de manière formelle les résultats intermédiaires des différentes étapes du processus de conception. La maintenance de la continuité du modèle est une caractéristique importante puisqu'elle facilite d'une part la conception sans erreur (pas de déformation ou perte d'information liée à une transcription de modèle) et

d'autre part améliore la traçabilité lors de la vérification et la validation du système conçu.

- Modèle unifié pour le matériel et le logiciel

Une autre caractéristique essentielle du modèle de représentation interne est qu'il doit être interprété indifféremment comme logiciel ou matériel ("modèle unifié"). Une telle représentation unifiée des parties logicielles et matérielles au sein d'un même modèle permettra d'envisager le partitionnement de manière efficace ainsi qu'un développement conjoint qui élimine les problèmes d'intégration des différentes parties du système conçu. Le besoin d'unification du processus de développement est donc nécessaire non seulement pour spécifier sous le même modèle des sous- systèmes aux implémentations différentes (matériel ou logiciel), mais aussi pour unifier les diverses activités nécessaires à travers les différentes phases de développement sous la même représentation interne (ce qui correspond au concept de "conception unifiée").

- Orientation du modèle vers un domaine précis

Chaque modèle de représentation repose sur un modèle d'exécution qui spécifie le flot de données et/ou le flot de contrôle. Selon la classification faite dans [Var 02], le modèle peut appartenir soit à la classe des modèles développés initialement pour les systèmes dominés par le contrôle et qui ont été étendus pour inclure le flot de données (noté MCD), ou à la classe des modèles développés initialement pour les systèmes orientés flot de données et qui ont été étendus pour inclure le flot de contrôle (noté MDC), ou enfin à la classe des modèles développés spécialement pour supporter une combinaison aussi bien du flot de contrôle que du flot de données (noté $M^{1}b$). Partant de l'évidence que le meilleur modèle est celui qui traduit le mieux les traitements de l'application du système, la connaissance de l'orientation du modèle est donc un facteur très utile pour mieux guider le choix du modèle le plus approprié.

- Capacité du modèle

Capacité du modèle à offrir des propriétés telles que: la puissance de modélisation (concurrence "parallélisme", communication, données et temps...), les possibilités d'analyse (possibilité de simulation, vérifiabilité, disponibilité d'outils...), la possibilité de synthèse (implémentation en matériel et en logiciel, disponibilité d'outils et méthodes

supportant le modèle, capacités d'automatisation...), le support de la complexité, l'abstraction et la hiérarchie.

4.2 Exploration de l'espace de conception du SoC

Tout au long du processus de conception et en particulier lors de l'étape de partitionnement matériel/logiciel, on cherchera le compromis matériel/logiciel le mieux adapté aux critères de performances et de coût visés (i.e la solution architecturale qui satisfait d'une façon optimale les contraintes imposées). Ces compromis (solutions architecturales) ne peuvent être examinés qu'à travers une exploration des différentes alternatives de conception. Cependant, étant donné la complexité des systèmes mixte matériel/logiciel, et l'éventail des possibilités offertes par la technologie, le nombre des alternatives de conception à examiner est extrêmement grand. Face à la taille exponentielle de l'espace des solutions et pour ne pas être fastidieux, le choix du meilleur compromis matériel/logiciel doit être le plus automatisé possible (partitionnement automatique) avec une exploration efficace de l'espace des solutions architecturales. Cette exploration, qualifiée d'efficace, doit permettre la recherche de solution optimale (meilleur compromis matériel/logiciel) dans un délai raisonnable à l'échelle humain sans parcourir pour autant l'ensemble de l'espace des solutions [Bag 02']. Ceci n'est possible qu'à travers la restriction, au moyen de contraintes, de l'espace des solutions à explorer à l'ensemble des solutions qui sont correctes vis à vis des contraintes imposées, en particulier en coupant les fausses pistes menant à des résultats sans intérêt [Nee 00]. L'objectif étant de réduire considérablement le nombre de solutions à examiner tout en augmentant la probabilité de trouver la solution optimale.

4.3 Réutilisation aisée

Il est généralement admis que pour appréhender la complexité grandissante des systèmes matériel/logiciel, il faut augmenter les possibilités de réutilisation aussi bien des applications que des architectures. Pour ce faire, la tendance des nouveaux environnements de conception conjointe consiste à adopter un modèle de construction basé sur une séparation nette entre les deux vues : fonctionnelle (algorithmes de l'application), architecturale (architecture matérielle d'implémentation) dès les premières

étapes du cycle conception- implémentation. Ce modèle de construction particulier (appelé "Y-chart approach") repose sur un environnent permettant de différencier la spécification des algorithmes décrivant le comportement de l'application, de la spécification de l'architecture supportant leur implémentation et de la spécification de l'étape d'implémentation proprement dite de l'application sur une architecture particulière. Cette séparation des modèles autorise aussi bien le portage des algorithmes et architectures que leur réutilisation : une même application peut être réutilisée sur une nouvelle architecture, ou être transformée et réimplémentée sur une architecture existante. Ceci favorise une exploration efficace de l'espace des implémentations possibles de l'algorithme sur l'architecture et par là même une réduction considérable du délai de la recherche de la meilleure implémentation (l'adéquation algorithme-architecture) ainsi qu'une arrivée plus rapide du produit final sur le marché.

4.4 Utilisation de règles et de méthodes formelles

La complexité croissante des systèmes conçus par des approches de conception hétérogène rend de plus en plus souhaitable, voire indispensable, l'utilisation de règles et de méthodes formelles tout au long du processus de conception et pas uniquement pour la vérification/validation à posteriori. L'objectif étant d'offrir des cadres formels qui permettent d'effectuer des techniques d'analyse statique de correction et de preuve formelle de propriétés ainsi que des simulations mixtes (logiciel/matériel) à différents niveaux d'abstraction du cycle de conception -implémentation assurant ainsi une validation continue tout au long de la conception. Il est clair que l'on ne construit pas un système pour vérifier ensuite qu'il fonctionne bien; on le construit essentiellement en ayant en permanence à l'esprit, depuis le début de sa conception jusqu'à sa réalisation finale, qu'il doit fonctionner suivant certains paramètres ou lois et cette propriété est constamment "prouvée" et totalement "intégrée" au processus de réalisation à tous les niveaux. C'est pourquoi, la seule méthode raisonnable pour atteindre ces objectifs est de recourir à des approches de conception basées sur des méthodes formelles permettant d'effectuer des analyses statiques de correction, indispensables pour assurer un développement rigoureux et systématique depuis la spécification de haut niveau jusqu'à la génération automatique ou la réalisation des programmes/circuits qui les implémentent.

4.5 Lien avec les outils de synthèse matérielle et logicielle

Pour assurer une exploitation automatique, et profiter de la technologie existante et très performante dans le domaine de la compilation et de la synthèse de circuits, il faut offrir des systèmes de conception capable de s'interfacer facilement aux principaux outils de CAO et compilateurs pour des machines mono ou multi- processeurs. Ce choix est imposé pour diverses raisons et considérations (maturité indiscutable de ces outils créés pour résoudre des problèmes bien précis, leur maintenance et amélioration constante, etc.).

5. Flots de conception des systèmes sur Puce

Plusieurs travaux ont identifiés la discontinuité et l'écart existant entre l'algorithme, le système, l'architecture, le matériel et la conception physique mais ils n'ont pas toujours contourné ce problème d'une façon complète et efficace. Certains de ces travaux ont pour but la réduction du gap entre l'algorithme et la conception de l'architecture en se basant sur des outils de synthèse en C/C++ [Cat], [Sem 01], [Wak 00]. Toutefois, ils se sont focalisés sur l'implémentation des blocs matériels automatiquement à partir d'une spécification de haut niveau. Une réponse partielle au défi de conception des systèmes multiprocesseurs sur puce était de développer des environnements qui regroupent plusieurs outils de conception [Bag 02], [Jan 02], [Lee 06], [Sys'], [Sor 05]. Par exemple un environnement de conception au niveau système et d'exploration d'architecture pour les MPSoCs était défini à partir du langage SDL [Bag 02]. EPICURE [Jea 06] est un autre environnement qui montre bien que la coopération entre des méthodologies complémentaires et des outils d'aide à la conception associés à des architectures appropriées peut améliorer de manière significative la productivité du concepteur, particulièrement dans le contexte des architectures reconfigurables. HyVisual [Lee 06] est utilisé pour la modélisation, la simulation, et la conception des systèmes embarqués à temps réel concurrents.

Néanmoins, nous ne pouvons pas présenter les flots et les outils de conception d'une façon exhaustive. Pour cela nous avons décidé de détailler respectivement deux outils académiques et un outil commercial.

- Ptolemy (outil assez mature sur lequel beaucoup de travaux sont déjà effectués).

- MAGIC (outil basé sur Matlab/Simulink, similaire à notre travail objet de cette thèse).
- System Studio de Synopsys.

5.1 Ptolemy/PtolemyII

PtolemyII [Lee 03], [Jan 01] est le successeur de Ptolemy 0.7 (appelé aussi "Ptolemy Classic" [Pto], [Liu 01], [Buc 94]) dont il reprend de nombreuses caractéristiques. Cet outil de recherche, développé à l'université de Californie, à Berkeley, sous la direction du Prof. Edward A. Lee, propose un environnement hétérogène de modélisation, de simulation et de construction d'applications concurrentes, dédié aux systèmes embarqués réactifs. Cet environnement de conception conjointe vise la conception des systèmes embarqués, spécifiquement les systèmes qui combinent plusieurs technologies différentes.

Ptolemy supporte la construction et l'interopérabilité de modèles exécutables selon des modèles de calculs très variés en autorisant l'imbrication hiérarchique des modèles. La décomposition récursive d'un système complexe en sous-systèmes réalisés par des modèles de calcul peut être hétérogène, c'est-à-dire que chaque décomposition peut avoir un modèle de calcul différent. Une application sous Ptolemy est donc décrite sous la forme d'un graphe d'acteurs (noeud d'exécution) reliés par des arcs ou l'interprétation des acteurs et des arcs diffèrent selon le "domaine" choisi. Ptolemy propose un certain nombre de domaines dont les plus importants sont : le modèle à événements discrets (DE"Discrete-Events"), le modèle à base de machines d'états finis (FSM Finite-State-Machine), le modèle à base de processus communicants (CSP Communicating Sequential Processes), le modèle à base de réseaux de processus synchrone (SDF Synchronous DataFlow) etc. Malgré la diversité des domaines offerte par Ptolemy, il offre cependant une infrastructure de spécification unifiée pour la partie matérielle et la partie logicielle sous forme de modèles de calcul ce qui facilite la migration de fonctions entre les deux types d'implémentation.

Le partitionnement dépend de plusieurs paramètres tels que le coût des communications, la surface ou la vitesse. Lors de la synthèse, le code généré pour la partie logicielle pourrait être du C ou de l'assembleur selon le processeur cible et du code SILAGE [Hil

93'] pour la partie matérielle. La simulation du système hétérogène a lieu une fois que la synthèse du logiciel, du matériel et des interfaces est faite. Les résultats de cette simulation permettent de vérifier si la conception est conforme à la spécification initiale. Cependant, à l'issue des étapes de synthèse du matériel et du logiciel, quelques estimations de performances sont faites afin de valider la conception. Les estimations concernent la surface, le chemin critique et l'utilisation de bus et de composants

L'architecture cible, dans Ptolemy, comprend une variété de processeurs qui peuvent avoir différentes configurations (mono-processeur, multiprocesseurs, architectures parallèles, etc.). Notons néanmoins que la spécification de cette architecture est entièrement à la charge de l'utilisateur, qui doit la simuler soit explicitement soit implicitement dans ses modèles.

Il est bon de noter que dans l'environnement Ptolemy, l'accent est mis surtout sur la simulation des systèmes. En effet, bien que Ptolemy soit un environnement fiable de simulation et de modélisation de systèmes complexes très hétérogènes à plusieurs niveaux de raffinements successifs, ses capacités de génération de code d'une implémentation exécutable en temps réel des systèmes cibles restent limités. Ptolemy peut générer du code C dont la qualité n'est souvent pas satisfaisante mais il peut rarement générer un code VHDL synthétisable, ce dernier n'est absolument pas optimisé. Le principal travail de fond du projet Ptolemy est la définition d'un cadre conceptuel pour la comparaison des différents modèles de calcul.

5.2 MAGIC

MAGIC est un environnement utilisé pour la conception des systèmes embarqués temps réels développés à l'institut de la technologie de Gerogia [Jan 02]. Il est orienté vers les applications de traitement de signaux caractérisés par un calcul puissant et des signaux ayant une large bande passante, comme le radar, le sonar et l'imagerie médicale. Il cible des plateformes muliprocesseur commerciales avec un réseau d'interconnexion. Le Tableau 2.1 montre les principales plateformes utilisées par cette méthodologie. L'entrée de ce flot de conception est en MATLAB. Elle est composée de l'application, et de l'ensemble des contraintes défini par le concepteur. Les outils intégrés dans MAGIC sont issus des environnements de Mathwork et de Innoveda et sont décrites comme suit.

Vendeur	Interconnexion	Processeurs
CSPI	Myrinet	I860, SHARC, PPC
Mercury Computer System	RACE, RACE++	I860, SHARC,PPC& Altivec
SKY Computers	SKYchannel	I860, SHARC, PPC &Altivec

Tableau 2.1: Les différentes technologies adaptées par l'outil MAGIC

- Mathwork DSP Workshop :

L'environnement MATLAB est utilisé pour la modélisation des algorithmes, et l'analyse fonctionnelle des applications de traitement des signaux de radar. Simulink est fortement attaché à MATLAB. Il permette à des expressions de MATLAB d'être employées explicitement dans des blocs de Simulink. La librairie DSP Blockset rend Simulink un environnement de prototypage rapide pour des applications de traitement de signal. Le RTW est un complément de Simulink utilisé pour générer du code C.

- EXCEL and Matlab Excel Link

L'Excel fournit le cadre requis pour la tabulation et l'analyse des contraintes de conception. Excel Link présente un service plutôt qu'un environnement. C'est un canal qui permet à Excel de copier les données dans MATLAB et de s'exécuter en restant dans Excel. Par conséquence le concepteur utilise un cadre de travail unique et simple.

-eArchitect

C'est un outil qui permet la modélisation des performances [Inn]. Il est utilisé pour l'exploration et l'analyse précoce des architectures pour les applications de traitement de signal implémentées sur un ensemble de plateformes déterminé.

MAGIC établit une continuité de modèle à travers les outils et le flot de conception et d'exploration. Cette continuité est réalisée en utilisant des middlewares de communication (MPI) et de computation (VSIPL : utilisés pour les plateformes embarqués temps réel ; [Vsi 00]) générés par Real –time workshop de Simulink. Ces

middlewares contribuent dans la détermination des retards 'Delay' des processus dans l'environnement de modélisation des performances afin d'estimer les performances de chaque configuration.

MAGIC accélère la conception et réduit le gap entre la spécification et l'implémentation. Il permet d'explorer l'architecture à un stade précoce avec une précision significative basée sur l'utilisation des middlewares. Cependant l'architecture ciblée est limitée à des architectures commerciales et à un ensemble de plateformes déterminés.

5.3 System Studio de Synopsys

Le « System Studio » [Syn 03] est un environnement de conception de haut niveau. Il est utilisé principalement pour la conception des architectures et des algorithmes au niveau système des applications SoC innovatrices. La conception des 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 consiste à bien choisir les processeurs, la logique spécifique, les bus, les mémoires et les périphériques afin d'aboutir à une utilisation plus efficace du silicium. La conception et l'intégration des éléments de matériel et de logiciel au niveau système, qui sont adaptées par l'outil System Studio, suppose l'utilisation d'un niveau d'abstraction plus haut que RTL. Le System Studio est un outil unifié qui s'occupe efficacement de tous les aspects importants relatif à la conception et à la vérification au niveau système du SoC :

- Capture et gestion de la 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 du système à plusieurs 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 de la langue 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 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 NCSim et Mentor ModelSim pour pouvoir cosimuler avec des blocs au niveau RTL décrits en VHDL ou Verilog.

6. Conclusion

Ce chapitre a été dédié à la description des systèmes objets de cette thèse, à savoir les systèmes multiprocesseurs monopuces. La définition appropriée au système monopuce est que c'est un système complet sur une seule pièce de silicium, résultant de la cohabitation de nombreuses fonctions déjà complexes en elles mêmes.

Après l'introduction des SoCs, leurs caractéristiques étaient détaillés : Hétérogénéité des composants constituants, embarquement et intégration dans un système complet, spécification suivant les applications implémentées, réduction de la latence, réduction du coût.

Le passage du monoprocesseur au multiprocesseur était ensuite évoqué tout en justifiant ce choix basé sur l'amélioration des performances des systèmes de plus en plus complexes. Les propriétés des systèmes multiprocesseurs étaient détaillées : Parallélisme, retards causés par les interconnections, temps de conception. Pour discuter après les différents domaines d'application des MPSoCs.

La troisième partie de ce chapitre a rappelé les différentes étapes de la conception des MPSoCs. Généralement la conception des systèmes sur puce part d'une spécification unique décrivant l'architecture. Ensuite, le partitionnement du système survient pour décomposer ce dernier en trois parties : Partie matérielle, partie logicielle et interface de communication. Ces trois parties obtenues sont ensuite vérifiées et validées avant de passer à la phase du raffinement et d'implémentation.

La quatrième partie a porté sur les caractéristiques requises des outils de conception des SoCs. Une telle liste de caractéristiques a servi pour indiquer les défis actuels et les domaines qui posent des problématiques et qui méritent des efforts dans les futurs travaux de recherche.

La dernière partie de ce chapitre a discuté les flots de conception des systèmes sur puce. En fait, plusieurs travaux ont identifiés la discontinuité et l'écart existant entre l'algorithme, le système, l'architecture, le matériel et la conception physique mais ils n'ont pas toujours contourné ce problème d'une façon complète et efficace. Deux outils de conception académiques et un outil commercial étaient détaillés dans cette partie pour illustrer d'une façon claire et concrète les divers travaux dans ce domaine.

CHAPITRE 3

METHODOLOGIE ET FLOT DE CONCEPTION MULTI-NIVEAUX DES SYSTEMES MULTI- PROCESSEURS MONOPUCE

Sommaire

1. Introduction

La grande difficulté en architecture, lors de la conception des systèmes multiprocesseurs monopuces, est de maîtriser la complexité. Toutefois, le point de départ est la spécification de l'application au niveau fonctionnel, et le point d'arrivée est l'architecture RTL finale (matérielle/logicielle). Le défi actuel est le progrès facile du modèle fonctionnel au modèle RTL, sous de fortes contraintes de temps et de qualité de conception. D'autre part, l'étude menée au 2ème chapitre de ce manuscrit, affirme qu'aucun environnement de conception des systèmes sur puce ne peut couvrir tous les domaines d'application ayant des fonctionnalités et des caractéristiques variées et diverses.

Nous visons essentiellement dans ce travail, les applications du traitement de signal qui ne cessent pas de se complexifier sans signe visible ou prévisible de saturation. Nous sommes intéressés par Matlab/Simulink comme un outil de conception de haut niveau qui permet l'exploration des algorithmes, l'analyse efficace et flexible des résultats de simulation en vue de corriger les modèles. Matlab/Simulink de MathWorks [Mat], est un environnement largement répandu et utilisé dans le domaine de traitement de signal.

Des travaux intéressants à partir de l'environnement Matlab\Simulink sont déjà effectués [Zer 04] [Xil], [Rea]. Ils ont été réalisés pour concevoir soit la partie matérielle de la puce [Zer 04], [Xil], [Dav 02] (c.-à-d. génération du HDL pour les ASICS et les FPGAs), soit la partie logicielle [Rea] qui cible un processeur unique (génération du code natif de C/C++, ou du code C pour le système d'exploitation et le DSP).

Donc, nous élaborons dans ce chapitre une méthodologie et un flot d'aide à la conception et à la validation multi niveaux des systèmes multiprocesseurs intégrés sur une puce. La méthodologie doit mener à la définition et à l'exécution d'un flot de conception complet, permettant la génération automatique du SoC-RTL synthétisable basé sur un langage de spécifications de très haut niveau, Matlab\Simulink.

Dans les sections suivantes, la méthodologie et le flot de conception des MPSoCs basés sur Matlab\Simulink sont détaillés. Quatre niveaux d'abstraction sont définis, permettant des raffinements et des validations progressifs et systématiques à travers ce

flot de conception. Dans la dernière section de ce chapitre les outils d'aide à la conception utilisés pour réduire les fossés entre les différents niveaux d'abstraction d'une façon aisée et efficace sont présentés.

2. Méthodologie et flot de conception proposés pour la génération des systèmes sur puce à partir d'une spécification fonctionnelle en Simulink.

La méthodologie et le flot de conception pour la génération des systèmes sur puce à partir de Simulink que nous proposons, sont représentés par la Figure 3.1 ci-dessous.

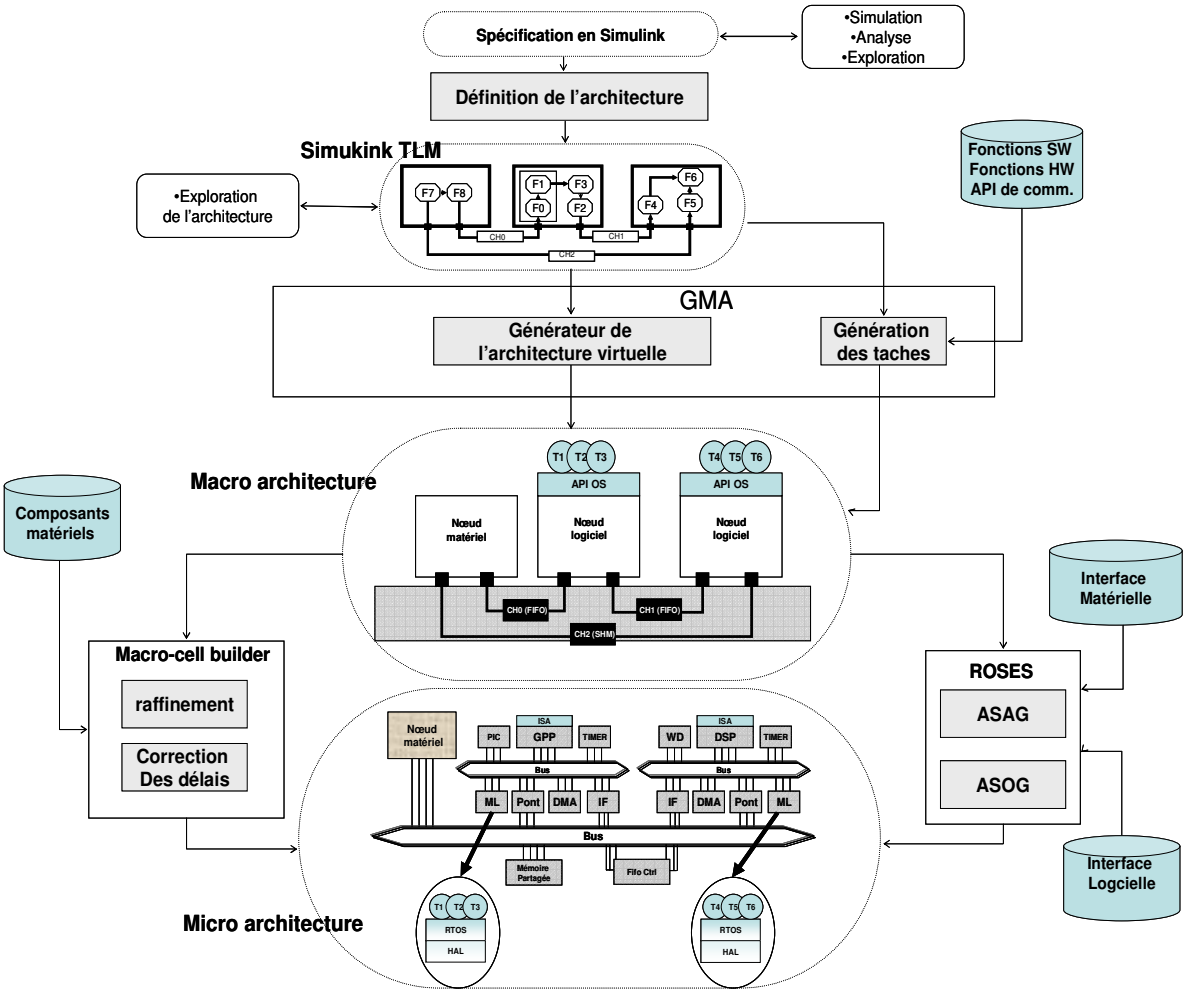


Figure 3.1: Flot de conception des MPSoCs basé sur l'environnement Simulink

La méthodologie de conception progresse systématiquement à travers des raffinements successifs suivant plusieurs étapes et niveaux. Le but est, donc d'arriver à une implémentation de l'application au niveau RTL et de cibler une plateforme hétérogène multiprocesseurs. Ainsi pour atteindre ce but, il faut réduire le grand fossé existant entre une description fonctionnelle et une description architecturale.

Dans ce flot, la description fonctionnelle est une spécification qui vise en premier lieu les applications de traitement de signal. Il est difficile de déterminer les coefficients et les paramètres convenables à ces applications au moyen des théories mathématiques toujours approximatives. Cependant, l'environnement Matlab\Simulink permet de modéliser et d'explorer les algorithmes de ces applications à travers des simulations itératives. D'autre part, la description architecturale que nous adoptons est une représentation abstraite au niveau macro architecture dans le langage Colif. Face à ce grand écart existant entre le niveau fonctionnel et le niveau architectural, il est pertinent de définir un niveau transactionnel intermédiaire dans Simulink afin de réduire ce gap. Ce modèle transactionnel permet de combiner l'algorithme et l'architecture. En principe, le TLM (Modélisation au niveau transactionnel) permet d'interagir les composants de calcul et de communication décrits à des niveaux d'abstraction différents en cachant les détails de calculs, de communications, et de leurs interfaces ou en les explicitant suivant le niveau d'abstraction décidé de chaque objet. Dans ce but, des règles d'abstraction sont définies pour modéliser l'architecture sous Simulink au niveau TLM. Ces règles seront détaillées dans la suite de ce manuscrit.

Le processus de prototypage élaboré par cette méthodologie a pour but l'automatisation de l'implémentation des applications de traitement de signal sur une plateforme hétérogène. Ensuite, un outil, le générateur de la macro architecture (GMA) est développé et plusieurs d'autres outils sont utilisés pour établir cette automatisation. L'entrée du (GMA) est le modèle transactionnel décrit en Simulink. La sortie est l'architecture virtuelle (VA) de l'application en Colif. Les tâches qui représentent le comportement de l'architecture virtuelle sont générées en SystemC. La synthèse de l'architecture est réalisée en utilisant l'outil ROSES [Ces 02], [Lyo 03], [Gau 01], et l'outil Macro_cell builder du groupe SLS [Zer 05]. L'architecture virtuelle résultante de

l'outil (GMA) est générée de façon à ce qu'elle soit adaptée aux différents outils intégrés dans ce flot. L'outil ROSES permet le raffinement des nœuds logiciels en générant des interfaces matérielles et des interfaces logicielles de l'architecture. Cependant le Macro-cell builder s'occupe du nœud matériel. Il permet le raffinement et la correction des retards dus à la différence du comportement entre le niveau fonctionnel et le niveau RTL du matériel. La validation du système se fait à travers des simulations progressives multi niveaux par des outils de cosimulation propres à chaque environnement.

3. Les niveaux d'abstraction utilisés dans le flot de conception des MPSoCs.

Les systèmes sur puce sont formés à travers un assemblage complexe de composants hétérogènes. Afin de simplifier le processus de conception, les concepteurs emploient généralement un certain nombre de modèles ou de descriptions intermédiaires. Les modèles intermédiaires [Cai 03] découpent la conception entière en plusieurs petites étapes de conception. Chaque étape a un objectif de conception spécifique. Comme ces modèles peuvent être simulés et estimés, le résultat de chacune de ces étapes de conception peut être validé indépendamment. Modéliser le prototypage d'un système aux différents niveaux est nécessaire, pour trois raisons :

(1) Fournir un niveau de conception convenable pour valider quelques détails d'implémentation pour un modèle représentant un système ou une partie d'un système.

(2) Fournir une vitesse de simulation suffisante pour valider et corriger le modèle.

(3) Faciliter et automatiser l'implémentation et la transposition d'une application sur des plateformes diverses.

Ainsi on peut définir généralement à travers notre flot de conception quatre différents niveaux d'abstraction (Figure 3.2): le niveau fonctionnel en Simulink, le niveau transactionnel en Simulink, le niveau macro architecture (architecture virtuelle), et le niveau micro architecture (Cycle accurate CA). Le but est de concevoir et de réaliser un système multiprocesseur sur puce.

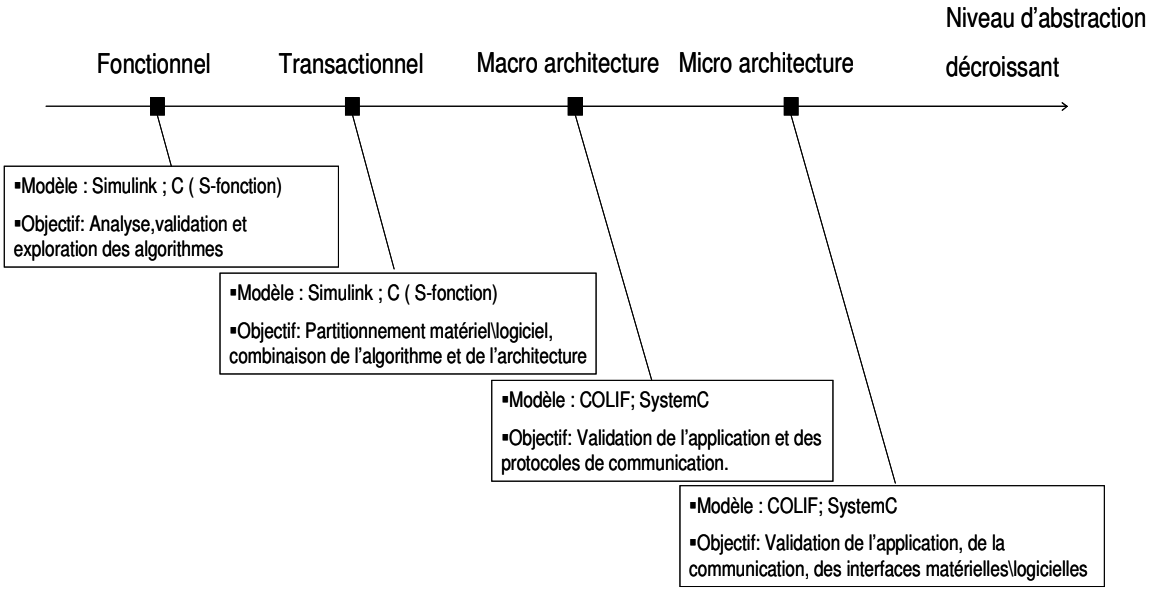


Figure 3.2: Les niveaux d'abstraction adoptés et leurs caractéristiques

3.1 Niveau Simulink fonctionnel

La conception d'un système monopuce commence toujours par la spécification. Cette dernière est une description qui contient toutes les fonctionnalités et les conditions que doit remplir l'application. Elle est indépendante de tous les détails d'implémentation du système. La modélisation fonctionnelle est un processus itératif difficile dans le domaine du traitement de signal. Cependant, le choix de Simulink facilite la modélisation et la validation des applications dans ce domaine. La modélisation fonctionnelle en Simulink est basée sur un assemblage de blocs paramétrables (éléments de la librairie) qui peuvent être implémentés avec des langages de haut niveau (Matlab, C/C++). La Figure 3.3 représente 9 unités fonctionnelles qui interagissent et échangent des données à travers des lignes d'interconnections dans l'environnement Simulink au niveau fonctionnel. Cependant, les blocs Simulink de la librairie sont variés et possèdent des caractéristiques différentes. Il existe des blocs fonctionnant à temps discrets et d'autres fonctionnant à temps continue. Le moteur de simulation utilise plusieurs solutionneurs et permet de résoudre des équations différentielles pour les systèmes continus. De plus les blocs de Simulink, existants et ceux définis par l'utilisateur sont conçus suivant plusieurs ordres de granularité.

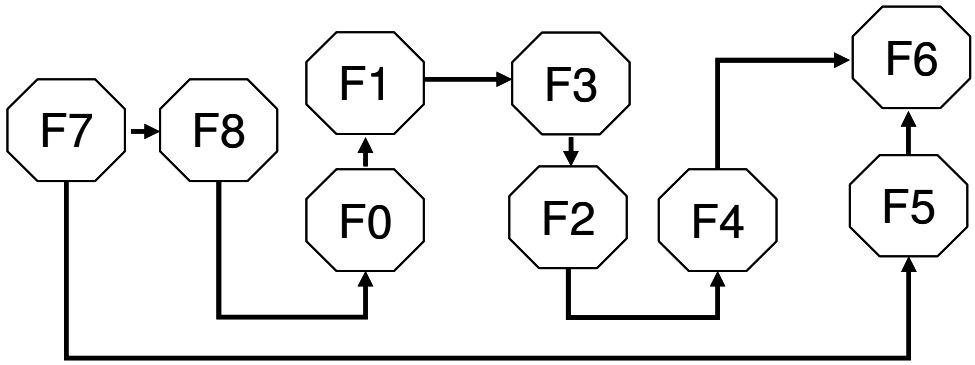


Figure 3.3: Spécification fonctionnelle en Simulink

Ils peuvent être des blocs élémentaires (+, *, -, /) comme ils peuvent représenter des fonctions typiques du domaine du traitement de signal ayant une granularité plus importante. Le parallélisme, la sérialisation et le branchement conditionnel peuvent être exprimés explicitement suivant plusieurs formes. Dans ce qui suit nous allons étudier ces choix pour définir le modèle et la configuration la plus adéquate pour faciliter la modélisation fonctionnelle et la génération d'un code multitâche, multiprocesseur.

3.1.1 Granularité des blocs du modèle fonctionnel

La complexité de l'application globale sera fragmentée en plusieurs modules de complexité plus accessibles. Par conséquence, les changements des contraintes fonctionnelles initiales seront vite modifiés sans changer l'ensemble globale. Dans Simulink, les blocs peuvent représenter des modules élémentaires ayant une granularité fine tels que l'additionneur et le multiplieur ou des blocs de traitement plus complexes ayant une granularité grossière tels que des filtres ou une FFT. Pour définir un modèle fonctionnel qui convient à notre objectif : la génération d'un code multitâche, multiprocesseur. Il est bien recommandé de modéliser les applications avec une granularité grossière. A cet ordre de grandeur, chaque bloc formant une partie de l'application peut être validé indépendamment du système et peut être réutilisé. La Figure 3.4 représente la fonction d'un filtre FIR suivant une granularité fine (gauche) et suivant une granularité grossière (droite). En fait, une description graphique est plus appropriée pour la description de l'architecture système qu'une architecture fine.

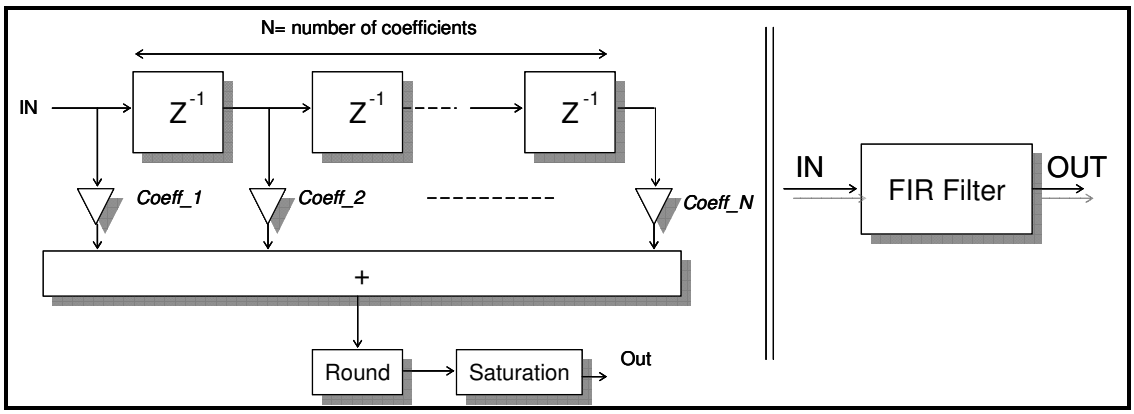


Figure 3.4: Une fonction modélisée avec une granularité fine et une granularité grossière

Toutefois, la modélisation fonctionnelle dans Simulink réalisée à partir des blocs élémentaires est un processus fastidieux. D'autre part, la modélisation suivant une granularité grossière accélère la modélisation fonctionnelle et facilite le partitionnement en diminuant le nombre des choix architecturaux.

3.1.2 Modularité et structure du modèle fonctionnel

L'ordre de grandeur grossière adopté par le modèle fonctionnel doit permettre la représentation explicite du parallélisme et du pipeline. Ces dernières propriétés sont exploitées en utilisant des modèles parallèles spécifiques pour effectuer des opérations de calcul intensif avec des contraintes de synchronisation. De telles opérations de calcul sont présentes dans les applications de traitement de signal et les applications multimédias (par exemple : Transformation DCT, …, etc.).

D'autre part, le simulateur de Simulink n'autorise pas les boucles de contre réaction dans le cas où un bloc de mémorisation n'est pas présent dans cette boucle. Ce problème sera résolu en utilisant un bloc de retard dans la boucle. De plus, il est très utile d'expliciter le branchement conditionnel entre les diverses fonctions typiques utilisées dans les applications de traitement de signal et les applications multimédias.

3.1.3 Ensemble de blocs Simulink utilisé dans le modèle fonctionnel

La librairie de Simulink comprend plusieurs types de blocs utilisés dans des domaines variés. Ces blocs peuvent fonctionner en mode continu et discret. Les signaux de communication entre ces différents blocs peuvent être des scalaires, des vecteurs et

des matrices. Ces communications sont modélisées par des arcs orientés. Un arc assure la connexion entre un bloc émetteur et plusieurs blocs récepteurs. Il transfert les événements à partir du port de sortie d'un émetteur pour alimenter un ou plusieurs blocs de réceptions. Dans notre travail, les blocs maintenus sont utilisés seulement en mode discret. L'utilisation des blocs qui supportent des signaux matriciels n'est pas autorisée. Le bloc virtuel 'Mux' qui permet de combiner plusieurs signaux vectoriels en un seul signal matriciel et le bloc 'DeMux' qui permet d'inverser cette opération ne sont pas adoptés par cette version du travail. Le branchement conditionnel est supposé établit par l'intermédiaire de la structure des blocs if-else (If-action subsystem) de la librairie de Simulink (Figure 3. 5). Cette structure fonctionne quand un événement produit par le bloc (if\else) est présent à l'entrée du contrôle du sous système. Dans le cas où le sous système n'a pas d'évènement de contrôle à son entrée, il ne produit pas des sorties. Les sorties de tous les sous-systèmes sont connectées au bloc 'merge'. Cependant, juste une seule sortie à un moment donné peut être présente à l'entrée du bloc merge. La Figure 3.6 représente la fonctionnalité d'un bloc de la librairie de Simulink utilisé dans la modélisation fonctionnelle. (I1, …, Im,….In) sont les entrées du blocs. (O1,…, Ok,…Oh) sont les sorties de ce bloc. Ce dernier correspond à des S-fonctions ou à des blocs prédéfinis. Les types basiques des variables utilisés par ces blocs de Simulink sont les suivants: booléen, double, single, int8, uint16, int32 et uint32.

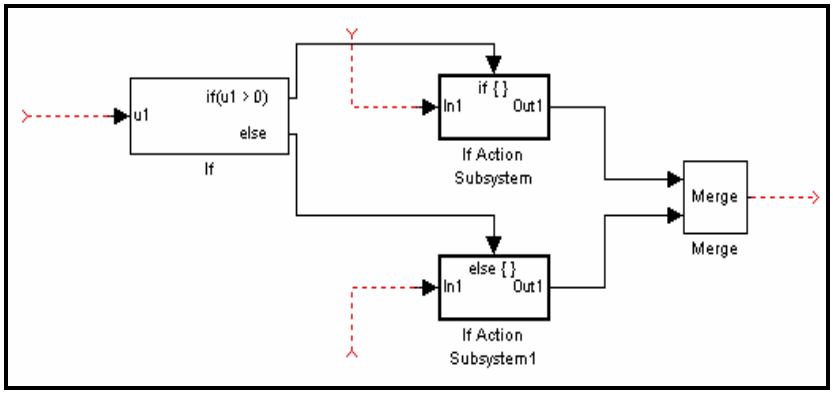


Figure 3. 5: La structure IF-else qui modélise le branchement dans le modèle fonctionnel de Simulink

Par défaut, les signaux sont considérés double sauf dans le cas où l'utilisateur définit explicitement le type, ou dans le cas de l'utilisation des blocs qui convertissent le type des données. Une erreur intervient dans le cas où deux types incompatibles sont utilisés dans une opération. Tous les blocs de Simulink ont des types polymorphiques. Le Tableau 4.1 illustre les types de données utilisés par quelques blocs de la librairie.

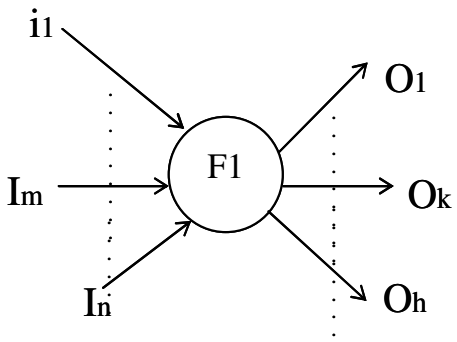


Figure 3.6: Représentation d'un bloc Simulink utilisé dans la modélisation fonctionnelle

Constant : α, α € SimNum

Adder : α x …x α → α, α € SimNum

Gain : α → α, α € SimNum

Relation : α x α → boolean, α € SimNum

Logical Operator: boolean x ……x boolean → boolean

Discret Transfer Function: double → double

Zero- Order Hold, Unit Delay: α→ α x α € SimT

InPort, OutPort : α → α, α € SimT

Tableau 4.1: Type de données de quelques blocs de Simulink

SimT représente touts les types de Simulink.

SimNum= SimT-{boolean}.

Par suite, les types de Simulink utilisés sont classifiés en plusieurs classes. (SimT, SimNum, {boolean}).

3.1.4 Méthodes utilisées pour implémenter le fichier Mex C (S-fonction)

S-fonction est un type de bloc existant dans la librairie de Simulink « **User- Defined Functions** », il offre à l'utilisateur la possibilité de concevoir son propre bloc en langage C. Chaque bloc constitue un modèle d'exécution indépendant et sera compilé séparément des autres blocs à l'aide d'un compilateur intégré à Matlab. Pour compiler un bloc S-fonction décrit en langage C, il faut éditer dans la ligne de commande de Matlab '**Mex « Nom du fichier »'**. Le bloc S-fonction est formé de deux parties (Figure 3.7): La première partie représente l'interface du bloc qui apparaît à gauche de la Figure 3.7. Le bloc possède une interface de dialogue. Le paramétrage de cette interface se fait graphiquement à travers une fenêtre de dialogue. Dans cette fenêtre il apparaît deux champs, le nom et les paramètres de la S-fonction. La deuxième partie est le code source de la S- fonction. C'est le fichier qui décrit le comportement, la computation, probablement en langage C. Il est présenté suivant un format bien précis. Dans ce fichier, appelé aussi 'C Mex file' (Si le code source est en Matlab on l'appelle 'M file'), le nombre, la taille et le type des ports d'entrée et de sortie du bloc sont défini. Le nom du fichier est « S-fonction.c **».** Il doit être indiqué dans le champ correspondant, dans la fenêtre de dialogue. Il doit être compilé dans Matlab pour mettre à jour les modifications dans le bloc (nombre de ports, nom du bloc).

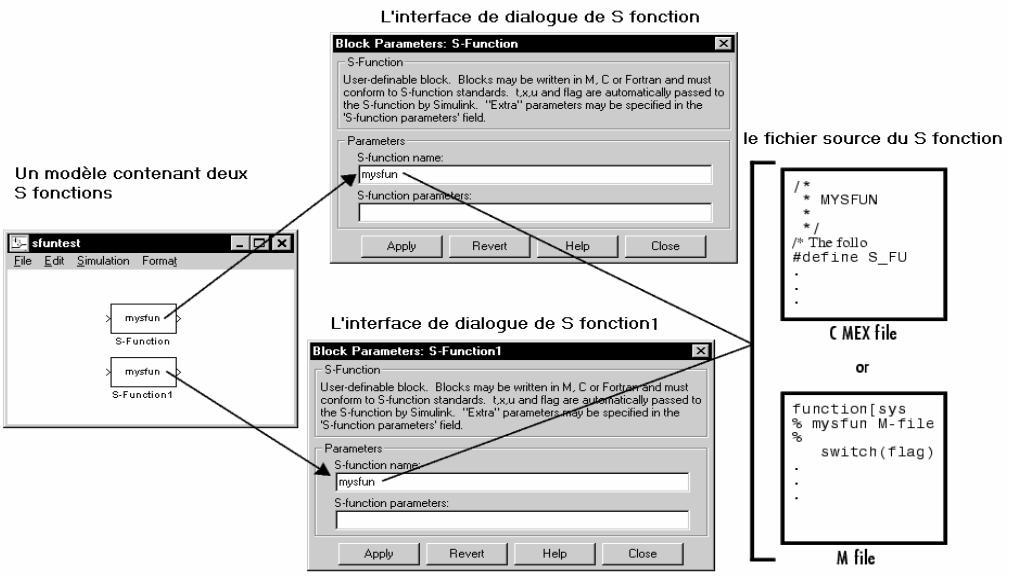


Figure 3.7: Configuration d'une S –fonction

Le fichier « S-fonction.c » est constitué de plusieurs méthodes définies dans Simulink. Ces méthodes sont traitées par le simulateur. Cependant nous utilisons dans notre modèle fonctionnel quatre méthodes parmi d'autres pour décrire le fichier Mex C. Ces méthodes sont les suivantes :

1- static void mdlInitializeSizes(SimStruct *S)

Dans cette fonction, le nombre et la taille des ports d'entrée et de sortie sont définis. Elle s'exécute une seule fois au début de la simulation pour initialiser les variables.

```
%%%
if (!ssSetNumInputPorts(S, nombre des ports)) return;
        ssSetInputPortWidth(S, numéro du port, taille du port);
%%%%
```

(`SimStruct`, est une structure de simulation qui contient des informations à propos du S-fonction).

2- static void mdlInitializeSampleTimes(SimStruct *S)

Dans cette fonction le temps d'échantillonnage du bloc est déterminé. L'option *« INHERITED_SAMPLE_TIME »* signifie que ce bloc prend le même temps d'échantillonnage que le bloc précédent.

```
%%%
ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
%%%
```

3- static void mdlOutputs(SimStruct *S, int_T tid)

Dans cette fonction :

- les signaux qui lisent et écrivent dans les ports d'entrée et de sortie sont modélisés.
- les données d'entrée sont traitées, et le calcul est réalisé.

```
%%%
InputRealPtrsType    u1= ssGetInputPortRealSignalPtrs(S, numéro du port); //entrée
real_T               *y1= ssGetOutputPortRealSignal(S,numéro du port); // sortie
%%%%
```

La communication se fait à travers l'affectation des primitives de communication par l'opérateur égale **« =».**

```
%%%
For (i=0; i< taille du port d'entrée ; i++)  valeur[i]=*u1 [i];        //   lire
For (i=0; i<< taille du port de sortie ; i++)  y1 [i]=valeur[i];        //    écrire
%%%
```

4- static void mdlTerminate(SimStruct *S)

Cette fonction s'exécute une fois la simulation est terminée.

A noter que nous pouvons utiliser des variables globales, propres à chaque bloc avant la fonction *mdlInitializeSizes (SimStruct *S).* IL y a plusieurs autres méthodes optionnelles utilisées dans la description des S- fonctions. Cependant, elles ne sont pas admises dans cette version du travail. Parmi ces méthodes, nous citons : *static void mdlDerivatives (SimStruct *S), static void mdlInitializeConditions (SimStruct *S), static void mdlUpdate (SimStruct *S, int_T tid),* cette dernière fonction est appelée à chaque pas de simulation, elle met à jour les états internes discrets du bloc Simulink quand ces états existent.

3.2 Niveau Simulink transactionnel (Simulink-TLM)

C'est un niveau transactionnel intermédiaire en Simulink utilisé pour faciliter la procédure de conception et d'implémentation des SoCs et pour combler l'écart existant entre le modèle fonctionnel (ni matériel, ni logiciel) et le modèle de l'architecture virtuelle.

A ce niveau les unités fonctionnelles homogènes seront assignées à des tâches logicielles ou à des IPs matériels.

Les choix de partitionnement sont pris à ce niveau comme suit:

1- Plusieurs unités fonctionnelles interconnectées sont fusionnées pour former une tâche logicielle ou un IP matériel.
2- Plusieurs unités fonctionnelles et tâches sont regroupées dans un sous système pour former un nœud logiciel.

3- Plusieurs unités fonctionnelles et IPs matériels seront regroupés dans un sous système pour former un noeud matériel.

Par rapport à la communication, les lignes d'interconnections au niveau fonctionnel sont simples. Ces lignes sont modifiées en des sous systèmes pour définir plusieurs types de topologies de communication entre les nœuds matériels\logiciels. (Cette topologie peut être point à point, multipoints, réseau de communication). De plus, ces interconnections peuvent être implémentés à ce niveau d'abstraction dans l'environnement Simulink pour valider plusieurs types et protocoles de communication. Le simulateur de Simulink assure le mécanisme de la communication et de l'ordonnancement, qui reste toujours implicites. L'importance de cette étape est de permettre l'exploration de l'architecture dans l'environnement Simulink. Une continuité de ce travail consiste à estimer les performances de l'architecture au niveau de Simulink, pour choisir l'architecture convenable. L'implémentation de ce modèle transactionnel fera partie du chapitre suivant.

La Figure 3.8 illustre le modèle de l'application au niveau transactionnel. Le partitionnement des unités fonctionnelles produit des tâches logicielles, des IPs matériels, des noeuds logiciels et des noeuds matériels.

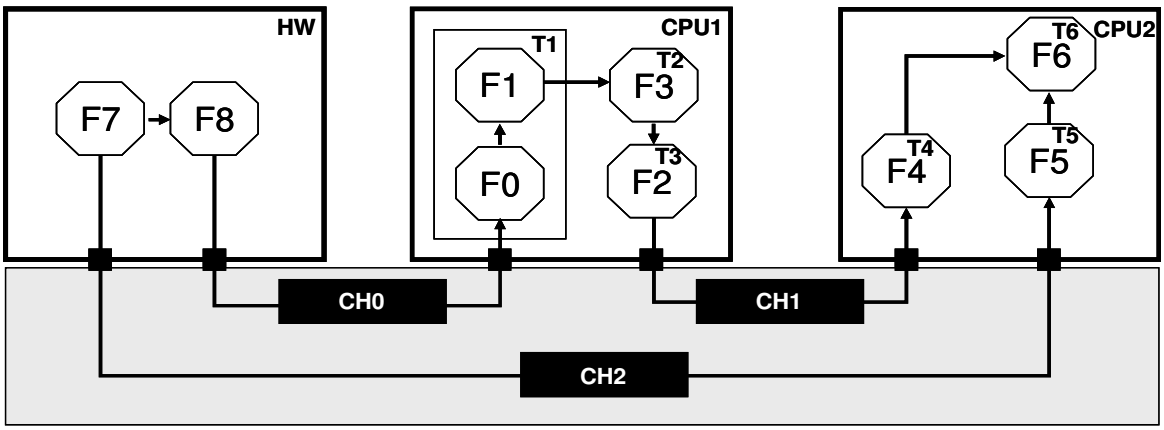


Figure 3.8: Exemple d'un modèle de l'application au niveau transactionnel en Simulink

3.3 Niveau macro architecture (Architecture virtuelle)

Ce niveau d'abstraction décrit l'architecture du système en excluant complètement tous les détails liés à la réalisation. Le système est réalisé sous la forme d'une architecture virtuelle formée par des composants virtuels. Un composant virtuel peut implémenter un ensemble de tâches logicielles ou un ensemble de fonctions matérielles mais sans aucune caractéristique précise sur le type du composant ou sur 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 composants virtuels travaillant concurremment et se communiquant via des canaux explicites et abstraits. Les canaux de communication utilisent des primitives transactionnelles définies par la norme TLM (Transaction Level Modeling) [Hav 02], 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. Ces primitives (API) ajoutées aux tâches, correspondent à une vision au niveau OS du logiciel applicatif. Le mécanisme de la communication se fait par une demande d'une donnée ou d'un bloc de donnée accomplie dans une seule transaction. Le temps est indiqué comme « temps passé » plutôt qu'un événement par horloge.

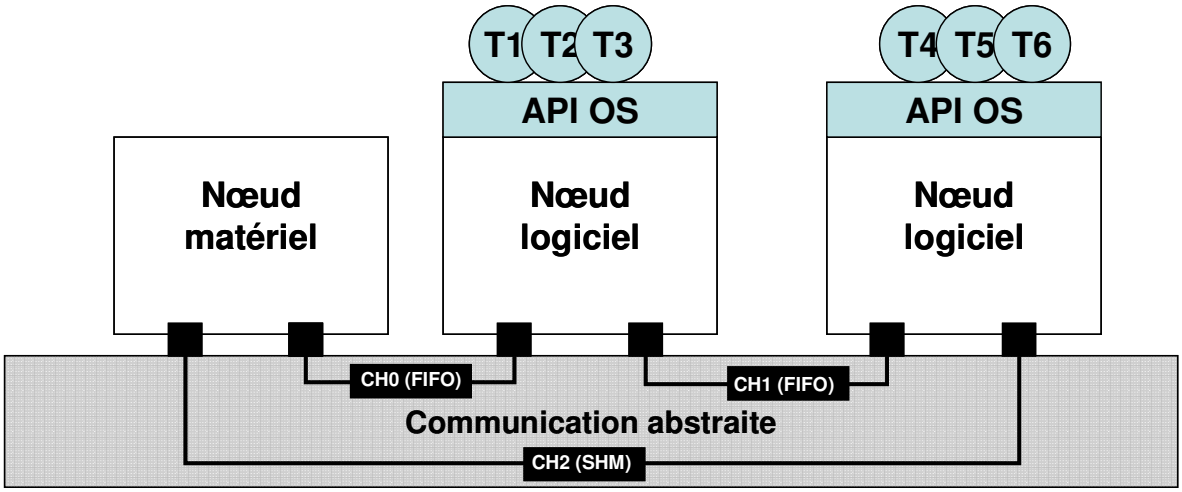


Figure 3.9: Exemple de l'architecture virtuelle

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 [CES 02].

La Figure 3.9 représente un exemple de l'architecture virtuelle. Des primitives de communications (APIs), sont ajoutées aux tâches logicielles pour abstraire le système d'exploitation.

3.4 Niveau micro architecture (CA Cycle accurate)

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 la micro architecture (Figure 3.10) la communication est précise au niveau cycle, tandis que les composants matériels restent fonctionnels et approximatifs par rapport au temps d'exécution. Les canaux virtuels transactionnels du 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. Les données sont transférées suivant l'ordre précis d'un protocole de bus. L'abstraction vient essentiellement des liens qui permettent de décrire un ensemble de 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 le code de l'application, le système d'exploitation et la couche HAL (couche d'abstraction du matériel). 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.

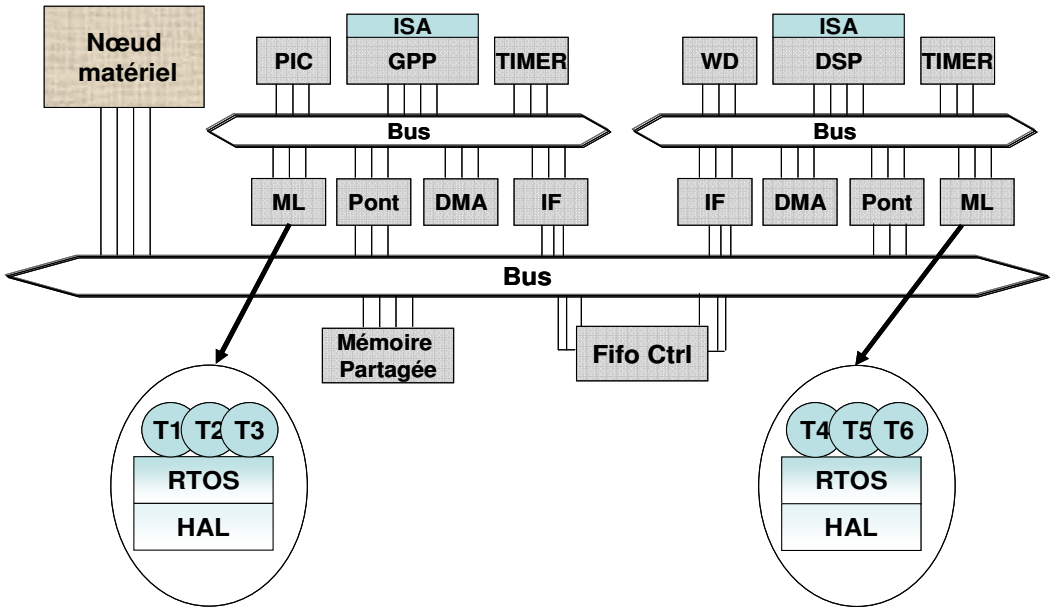


Figure 3.10: Exemple de la micro architecture

La validation du modèle de système à ce niveau est réalisée par la co-simulation du matériel et du 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.
- 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 pipe UNIX). Cependant, 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.

4. Les outils de conception utilisés dans le flot de conception des SoCs

La méthodologie proposée précédemment requiert l'interaction de plusieurs outils de conception. Ces outils s'articulent autour d'un modèle intermédiaire décrit avec le

langage Colif. Ce dernier, et les différents outils utilisés dans le flot de conception déjà proposé sont détaillés dans ce paragraphe.

4.1 COLIF

Colif est un langage qui permet de représenter un modèle intermédiaire orienté objet. C'est l'abréviation de (CO-design Langage Independent Format) [Ces 01]. La spécification en Colif décrit un système comme un ensemble d'objets interconnectés de trois types : les modules, les ports et les liens (en anglais : net). Un objet, quel que soit son type, est composé de deux parties. La première partie, nommée «entity», permet la connexion avec les autres objets. La deuxième partie, nommée «content», fournit une référence à un comportement ou des instances d'autres objets. Cette possibilité d'instancier des éléments permet le développement de modules, ports et liens hiérarchiques. La Figure 3.11 montre un modèle décrit en Colif. Les boîtes carrées représentent les modules, les petites boîtes foncées représentent les ports, et les lignes représentent les liens. Les fonctionnalités de chaque élément sont les suivantes :

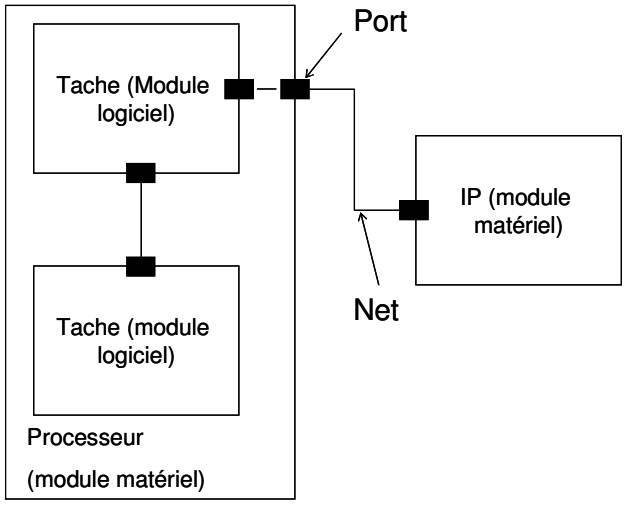


Figure 3.11: Un exemple de modules, de liens, et de ports

1. Le module représente un composant matériel ou logiciel. Son «entity» donne son type et les ports par lequel il communique. Son «content» peut être caché (boite noire), contenir des références à un comportement (ex : fichier C ou VHDL/matériel connu), mais peut aussi être un sous-système décrit sous forme de modules, ports et liens.

2. Le port représente un point de communication pour un module. Son «content» est composé de deux parties: l'une est en relation avec l'intérieur du module et l'autre avec l'extérieur. Ces deux parties peuvent être cachées, contenir des références à un comportement ou des instanciations de ports.

3. Le lien (Anglais: net) représente une connexion entre plusieurs ports. De la même façon, un lien peut contenir des références à un comportement ou contenir d'autres liens.

4.2 Environnement Simulink

Simulink est basé sur l'environnement Matlab. C'est l'un des outils le plus populaire utilisé pour la modélisation et la simulation des systèmes. La première version était réalisée en 1989 pour la conception et l'analyse des systèmes de contrôle. Aujourd'hui, cet outil est utilisé pour modéliser et simuler des applications appartenant à différents domaines (traitement de signal, système mécanique, électronique) dans un environnement temps réel. Simulink fourni une interface graphique pour la modélisation des systèmes dynamiques sous forme de schémas blocs, ainsi qu'une machine d'exécution permettant la simulation de ces modèles.

Ce logiciel offre de nombreuses fonctionnalités :

- une approche reposant sur les signaux,
- un grand nombre de modèles prédéfinis, sous forme de composants intégrés dans des boites à outils,
- un langage de programmation spécifique qui permet de définir de nouveaux composants,
- enfin, la possibilité d'encapsuler des modèles les uns dans les autres (hiérarchie de composants).

4.2.1 La communication entre les blocs de Simulink

Les communications entre les blocs d'un système décrit sous Simulink sont modélisées par des arcs orientés. Un arc assure la connexion entre un bloc émetteur et plusieurs blocs récepteurs. Cet arc réalise une fonction de transmission des données qui peuvent être une valeur ou un vecteur de taille défini. Un système Simulink peut utiliser l'environnement Matlab dans sa modélisation. Il peut alors communiquer des données à l'espace de travail de Matlab. L'échange de données entre Simulink et l'espace de travail Matlab peut se faire à l'aide de variables communes ou par l'intermédiaire des fichiers Matlab.

4.2.2 Moteur de simulation du modèle fonctionnel

L'environnement d'exécution dépend de la nature du solutionneur utilisé dans Simulink. Cet environnement permet l'échange des différentes valeurs de sorties entre les différents blocs. Deux phases d'exécution sont nécessaires pour réaliser cet échange : la phase d'initialisation et la phase d'exécution. La phase d'initialisation est faite pour tous les blocs en même temps et leur exécution suit un algorithme dépendant du solutionneur utilisé. Elle comprend les actions suivantes:

- Evaluation des expressions des paramètres des blocs.
- Mise à plat de la hiérarchie des modèles.
- Détermination de l'ordre de la mise à jour des blocs (phase de tri).
- Détermination des attributs des signaux et vérification de la concordance entre signaux.
- Détermination du temps d'échantillonnage pour les blocs n'ayant pas cette spécification.
- Allocation et initialisation de la mémoire utilisée pour sauvegarder les états et les sorties de chaque bloc.

La phase d'exécution comprend les actions suivantes :

- Le calcul successif des états et des sorties du système à des intervalles, du début de la simulation à sa fin.
- Le calcul du temps d'avancement dépendant du solutionneur.

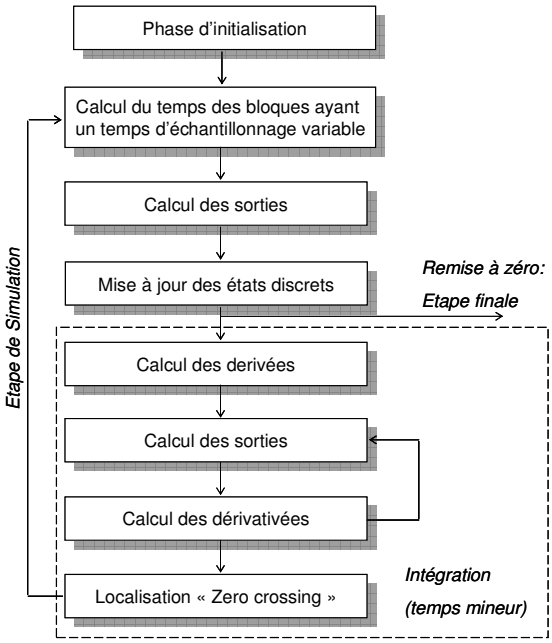


Figure 3.12: Ordonnanceur de Simulink

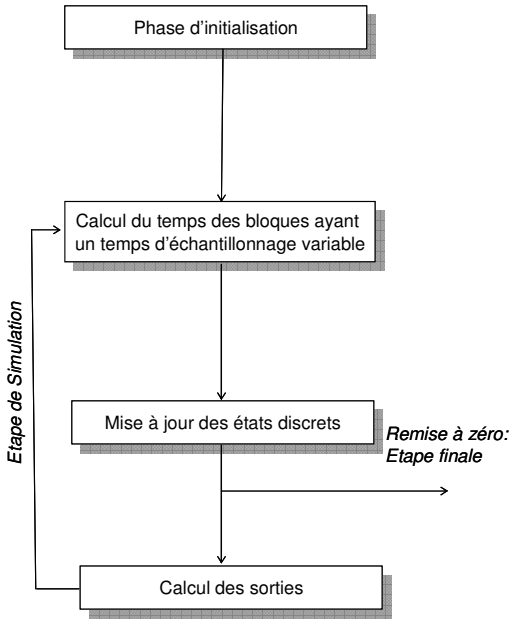


Figure 3.13: Ordonnanceur du modèle fonctionnel adopté dans Simulink

La Figure 3.12 représente l'ordonnanceur générique de l'environnement Simulink. Cependant, la Figure 3.13 illustre un cas particulier d'usage de cet ordonnanceur correspondant au modèle fonctionnel adopté dans notre travail. Dans la description fonctionnelle choisie, la hiérarchie n'est pas supposée utilisée. En effet, le concept de la hiérarchie sera défini dans un autre niveau d'abstraction et une sémantique précise lui sera attribuée. Nous modélisons dans ce travail uniquement les systèmes numériques. Donc les systèmes continus ne sont pas considérés dans cette spécification fonctionnelle. Pour la première phase d'exécution du modèle Simulink (l'initialisation), le temps d'échantillonnage est supposé discret. Le pas d'échantillonnage n'est pas variable comme dans le cas des systèmes continus. Ainsi, un solutionneur discret est utilisé le 'solver : fixed-step, discret'. Le calcul du temps d'avancement dans la phase d'exécution est invariable. De plus, le calcul des dérivées n'est pas exécuté par l'ordonnanceur de ce modèle parce que un solutionneur discret à pas fixe est utilisé.

4.2.3 Horloge de synchronisation du modèle fonctionnel

La plupart des blocs prédéfinis de la librairie de Simulink et les blocs définis par l'utilisateur possèdent un paramètre de configuration appelé temps d'échantillonnage (Sample time). Ce paramètre détermine l'horloge à laquelle le bloc est sensible. Ce paramètre peut être défini comme étant égal à la valeur '-1'. Cette valeur signifie que le bloc hérite la même horloge du bloc précédent, c'est-à-dire le bloc qui lui fournit les données. Si tous les blocs du système fonctionnent au même pas d'échantillonnage alors le système est mono fréquence (mono-rate). Cependant, dans le cas où les blocs fonctionnent à des horloges différentes, le système est multi fréquence (multi-rate). Dans notre modèle fonctionnel, nous supposons utiliser le système mono fréquence donc le paramètre du temps d'échantillonnage est fixé à '-1' pour les blocs prédéfinis et 'inherited sample time' pour les blocs définis par l'utilisateur. Le modèle fonctionnel, dans ce cas est sensible à une horloge abstraite qui synchronise le fonctionnement du système.

4.3 Le générateur de la macro architecture

La maîtrise de la complexité grandissante des systèmes multi processeurs sur puce consiste à accroître le niveau d'abstraction lors de la spécification. Ceci produit l'augmentation du fossé et la discontinuité du modèle entre le niveau de la spécification fonctionnel et le niveau qui représente l'architecture. Dans notre flot de conception, ceci se manifeste par le fossé présent entre le modèle spécifié en Simulink et le modèle au niveau macro architecture décrit en Colif.

Le générateur de la macro architecture permet d'automatiser le passage du modèle Simulink au modèle de l'architecture virtuelle en Colif. Ceci réduit le grand fossé et entraîne à établir une continuité entre les deux modèles, tout en accélérant la procédure de la conception des systèmes multiprocesseurs monopuce.

Le générateur de la macro architecture fera l'objet du chapitre 4 de ce manuscrit

4.4 Les outils de génération automatique des interfaces : ROSES

ROSES est formé d'un ensemble des outils développés au groupe SLS, il permet la génération automatique des interfaces logicielles/matérielles des systèmes monopuces.

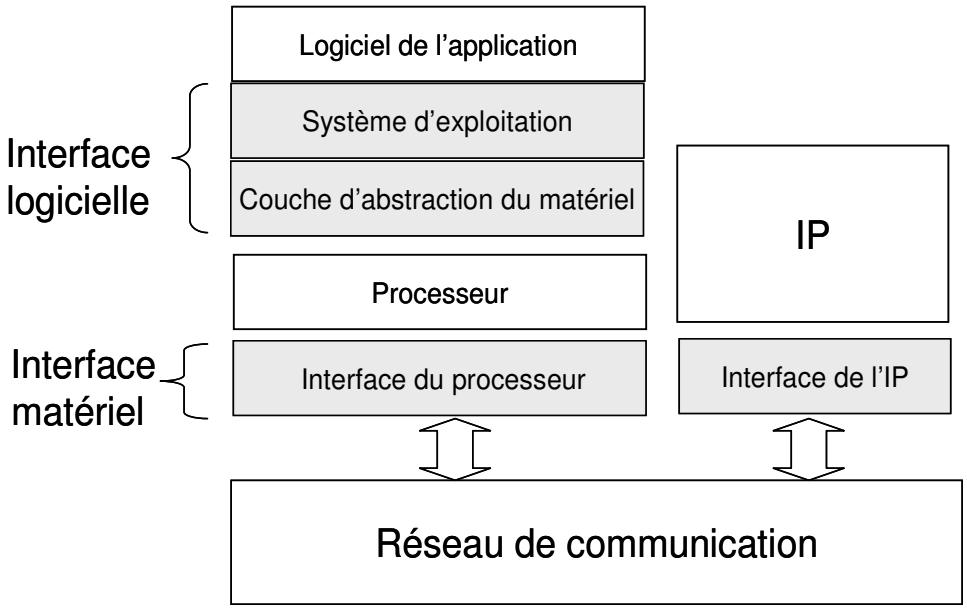


Figure 3.14: Interface logicielle/matérielle

L'interface logicielle/matérielle (Figure 3.14) générée est un adaptateur de communication entre les tâches exécutées par le processeur et les ressources matérielles. Plus précisément l'interface logicielle permet au code de l'application d'accéder aux ressources matérielles du processeur (typiquement pour faire des E/S) et l'interface matérielle permet au processeur d'accéder au réseau de communication. Les interfaces matérielles prises en compte par ROSES concernent les processeurs [Lyo 03], mais aussi la mémoire [Gha 03], et les composants matériels spécifiques [Gra 04]. Les interfaces logicielles sont les systèmes d'exploitation [Gau 01] et les couches de communications [Pav 04]. 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, ROSES permet une validation à plusieurs niveaux d'abstraction à l'aide des mécanismes de co-simulation [Nic 02].

Le principe le plus important dans ROSES est la conception d'un système complet par assemblages d'éléments [Ces 02]. 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 : cela consiste en un assemblage d'éléments de bibliothèque.

ROSES est composé principalement de trois outils : ASAG, ASOG, et Cosimx.

- ASAG est l'outil de génération des interfaces matérielles.
- ASOG est l'outil de génération des interfaces logicielles.
- Cosimx est l'outil de génération des interfaces pour la simulation.

La Figure 3.15 montre les outils de ROSES qui permettent la génération des interfaces à partir d'un système au niveau macro architecture vers une description au niveau RTL. L'entrée des outils de ROSES est une architecture virtuelle décrite en Colif.

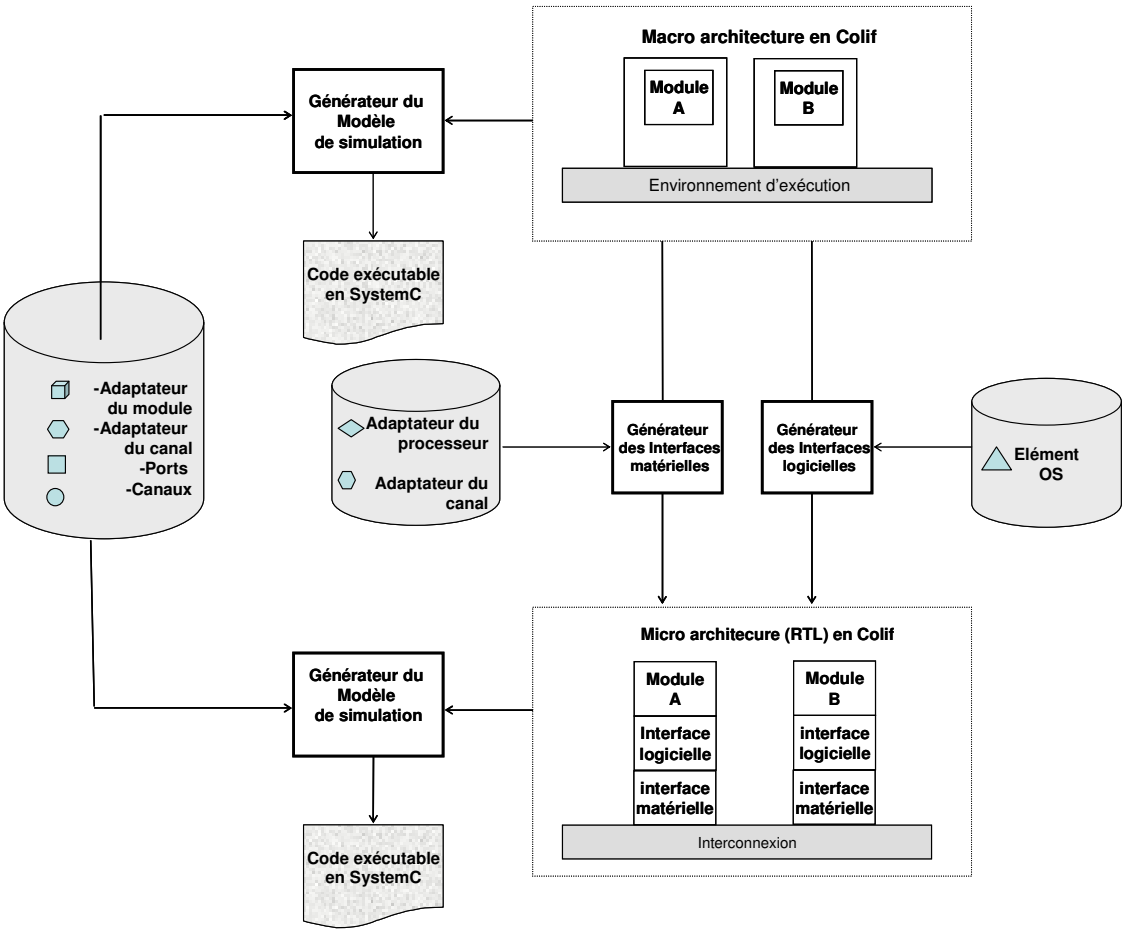


Figure 3.15 : ROSES

4.4.1 ASOG

L'outil ASOG [Gau 01] génère l'interface logicielle qui consiste en un système d'exploitation minimal et la couche d'abstraction du matériel à partir de paramètres donnés dans les ports. Les paramètres des ports de tâches décrivent les services de communication demandés et requis par la tâche. Les paramètres de ports internes indiquent l'implémentation de l'interface logicielle.

Les quatre actions exécutées par ASOG pour générer l'interface logicielle sont : analyse de l'architecture, lecture de la bibliothèque, sélection des composants, et expansion des composants. A l'étape de l'analyse de l'architecture, l'outil recherche le matériel utilisé (le processeur), les services requis (paramètres des ports de tâche), et l'implémentation de ce service (paramètres des ports internes du processeur). Ensuite, l'outil sélectionne les éléments de bibliothèque nécessaires pour construire ces services. Enfin, les éléments de bibliothèque sont paramétrés pour obtenir le code de l'interface logicielle.

4.4.2 ASAG

L'outil ASAG [Lyo 03] 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, nous obtenons une description Colif au niveau RTL de l'architecture de l'interface et de celle de l'architecture locale du processeur. Nous obtenons aussi le code synthétisable VHDL ou SystemC de l'architecture.

4.4.3 COSIMX

L'outil Cosimx est utilisé pour valider les systèmes décrits à plusieurs niveaux d'abstraction [Nic 02]. Il permet de rendre exécutable la description Colif du système. L'exécution du système avant génération des interfaces permet de valider le comportement des tâches, des IPs ainsi que le choix des services de communication. Cosimx offre aussi la possibilité de tester le fonctionnement du système après le raffinement partiel ou complet du système (cosimulation).

4.5 Macrocell builder

L'outil Macrocell builder [Zer 05], permet le raffinement des modèles fonctionnels, pour générer un modèle RTL matériel. Ce raffinement est réalisé en corrigeant les retards dus à la différence du comportement du modèle fonctionnel et du modèle RTL. Ces deux modèles sont conçus à partir de deux librairies de blocs basics prédéfinies et paramétrables. (Librairies fonctionnelles et RTL). La méthodologie mise en oeuvre par cet outil est décrite dans la Figure 3.16 ci-dessous.

Après la génération de l'architecture virtuelle par l'outil GMA. Le Macro cell builder intervienne pour traiter les Macro blocs matériels (nœuds matériels) qui sont formés par plusieurs IPs fonctionnels. Le processus de raffinement conserve la même architecture et remplace chaque IP fonctionnel par l'IP RTL correspondant.

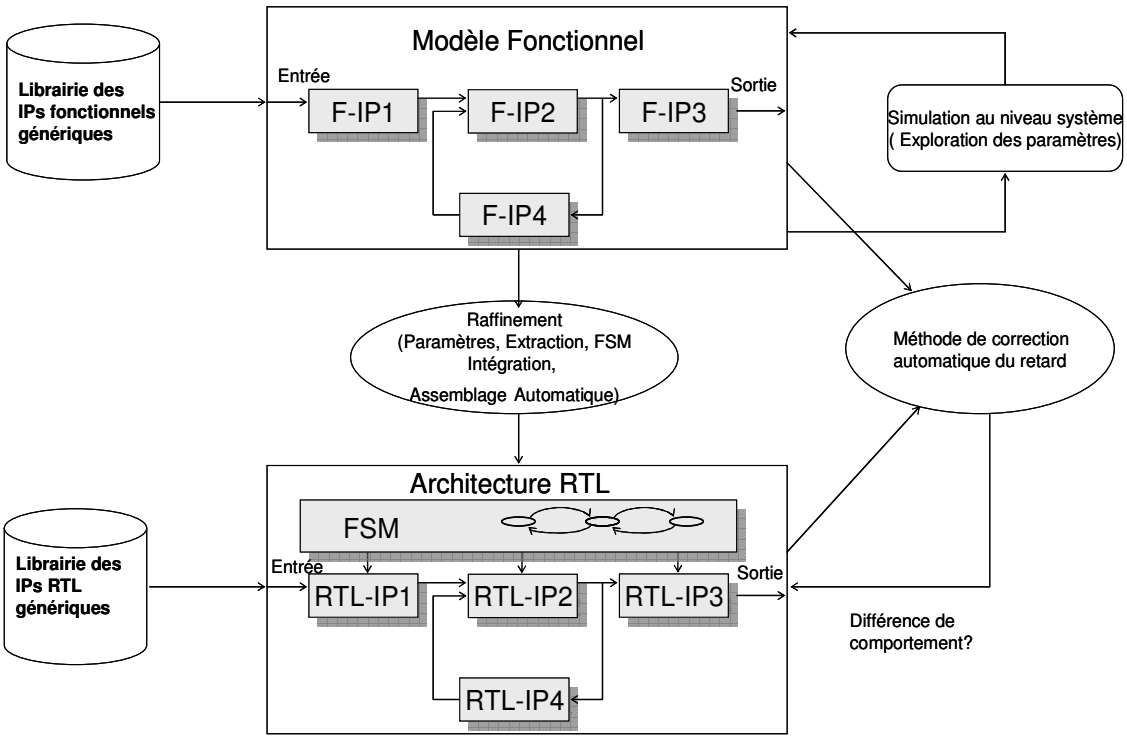


Figure 3.16: La méthodologie de l'outil Macro cell builder

Par conséquence, pour générer l'architecture matérielle RTL, les valeurs des paramètres des IPs sont extraites à partir du modèle de l'architecture fonctionnelle et sont utilisées pour instancier les IPs RTL prédéfinis, décrits en un langage matériel synthétisable (i.e. VHDL, SystemC). Ensuite le retard est corrigé par insertion des registres ou bien par insertion d'une machine d'état fini FSM. Le modèle RTL généré est approprié pour la synthèse logique.

5. Conclusion

Dans ce chapitre, nous avons présenté une méthodologie et un environnement pour la conception, la validation et le prototypage rapide des systèmes multiprocesseurs sur puce pour les applications de traitements de signal. L'approche présentée prend en considération la spécification au niveau système, la validation multi niveaux, l'exploration d'algorithme, le raffinement et le prototypage.

Notre flot de conception a permis de définir quatre niveaux d'abstraction pour représenter l'application et l'architecture. (1) Le niveau fonctionnel en Simulink qui décrit toutes les fonctionnalités et les conditions que doit remplir l'application. En restant indépendant de tous les détails d'implémentation du système. Le choix de Simulink a facilité la modélisation et la validation des applications à ce niveau d'abstraction. Par conséquence, une partie de ce chapitre était consacrée à définir le modèle fonctionnel et la configuration la plus adéquate dans l'environnement Simulink. Le but était de faciliter l'intégration de ce modèle dans le flot entier, ce qui a permis par la suite de générer une plateforme multiprocesseur et un code multitâche. (2) Le niveau transactionnel en Simulink était décrit brièvement, il fera partie du chapitre suivant où il sera plus détaillé. (3) le modèle au niveau macro architecture (architecture virtuelle) qui décrit le système suivant une architecture virtuelle formée par un ensemble de composants virtuels travaillant concurremment et se communiquant via des canaux explicites et abstraits. (4) Le niveau micro architecture (Cycle accurate CA) où l'implémentation des protocoles de communication et les composants de matériel et de logiciel utilisés dans les sous systèmes de l'application est réalisée.

La dernière partie de ce chapitre a évoqué les outils utilisés dans notre flot de conception. (1) Colif, un langage qui permet de représenter un modèle intermédiaire orienté objet. (2) Le générateur de la macro architecture qui permet d'automatiser le passage du modèle transactionnel Simulink au modèle de l'architecture virtuelle en Colif. Cet outil sera détaillé dans le chapitre suivant. (3) ROSES, formé d'un ensemble des outils permettant la génération automatique des interfaces logicielles/matérielles des systèmes monopuces. (4) MacroCell builder qui permet le raffinement des modèles fonctionnels, pour générer un modèle RTL matériel.

Ainsi, à la fin de ce chapitre nous avons une vue globale sur la méthodologie et le flot de conception proposés pour concevoir aisément et efficacement des MPSoCs dédiés aux applications de traitement de signal.

Le chapitre suivant rappellera le gap existant entre le niveau fonctionnel et le niveau macro architecture et les solutions adoptés pour réduire ce fossé.

CHAPITRE 4

GENERATION DE L'ARCHITECTURE VIRTUELLE À PARTIR DU MODELE TRANSACTIONNEL EN SIMULINK

Sommaire

1. Introduction

Un moyen de maîtriser la complexité grandissante des systèmes sur puce et des applications de traitement du signal et multimédia embarqués est d'élever le niveau d'abstraction de la modélisation. Cependant, un grand fossé sera créé entre le niveau fonctionnel et le niveau architectural. Ce fossé a pour effet de retarder la procédure de mise sur le marché des produits : un vrai problème économique !

La Figure 4.1 représente une partie du flot de conception proposé au 3ème chapitre. Elle illustre le fossé et évoque une discontinuité entre le modèle fonctionnel en Simulink et le modèle du prototype virtuel en Colif. Un modèle décrit à un niveau transactionnel intermédiaire en Simulink sera introduit dans ce flot de conception. Ce modèle permet de réduire le gap produit par l'élévation du niveau d'abstraction et d'établir une continuité entre les deux modèles écartés.

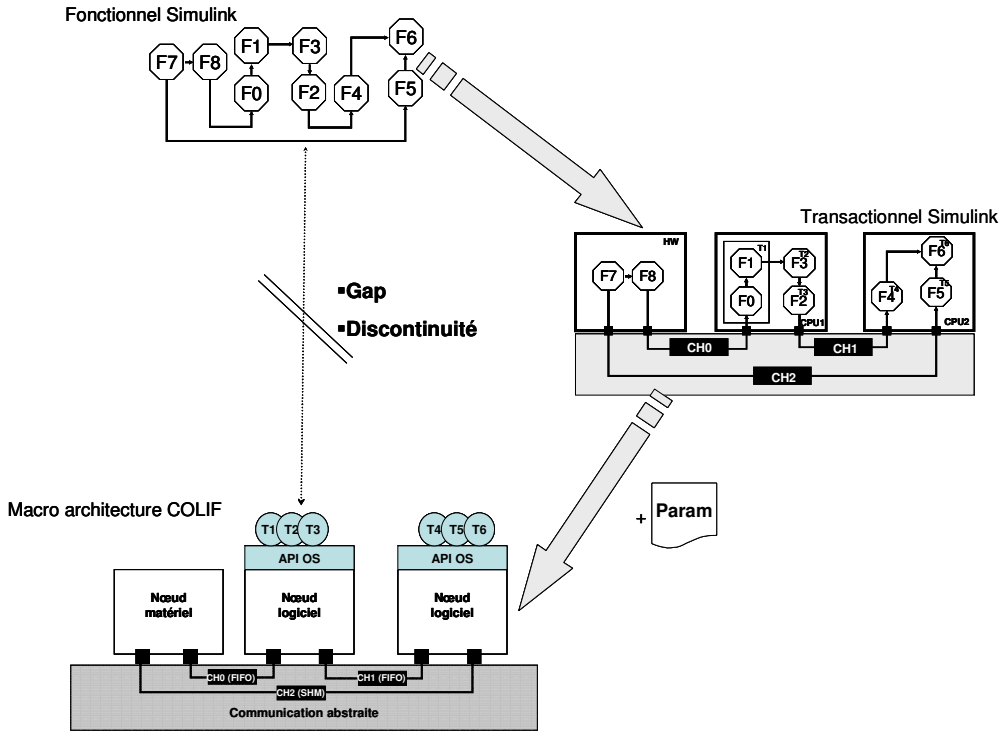


Figure 4.1: Introduction du modèle transactionnel intermédiaire

Dans ce chapitre, nous expliquons la modélisation transactionnelle Simulink proposée dans le flot de conception des MPSoCs. Ensuite, nous détaillons les étapes de l'outil qui transpose l'application en un prototype virtuel.

2. La modélisation au niveau transactionnel en Simulink

Le niveau transactionnel intermédiaire proposé dans le flot de conception des MPSoCs est indispensable pour réduire le fossé existant entre la description fonctionnelle en Simulink et la description de l'architecture virtuelle.

En effet, Simulink n'est pas fait pour modéliser des architectures multiprocesseur. Cependant, des règles de conception sont imposées pour définir ce niveau intermédiaire qui constitue l'entrée de l'outil de génération de la macro architecture. La modélisation à ce niveau exige que le concepteur connaisse bien son architecture (Les parties matérielles, les parties logicielles, les tâches exécutées sur chaque nœud logiciel, la topologie de communication, .. etc).

Au niveau fonctionnel de Simulink, nous supposons qu'il n'y a pas de hiérarchie et que tout le système se trouve à un même niveau. Les blocs sont interconnectés par des lignes simples unidirectionnelles partant d'une source et visant plusieurs récepteurs.

Les concepts de la modélisation transactionnelle en Simulink consistent à ajouter des modules de communication et à établir des partitionnements matériels/logiciels pour combiner l'algorithme et l'architecture. L'outil de génération de l'architecture virtuelle requiert une description Simulink transactionnelle structurelle à l'entrée. Cependant, ce modèle transactionnel peut être raffiné dans l'environnement de Simulink. L'implémentation du comportement des modules de communication permet de valider les protocoles de communication et l'interaction entre les différents modules.

Les éléments utilisés pour décrire l'architecture, sont les composants de calcul et les composants de communication qui sont détaillés par la suite.

2.1 Les composants de Calcul

Les composants de calcul utilisés au niveau transactionnel en Simulink sont constitués par des parties matérielles et logicielles. Les parties matérielles peuvent êtres des IPs élémentaires ou bien des macro blocs matériels hiérarchiques qui encapsulent des IPs matériels élémentaires. Les parties logicielles sont des tâches logicielles encapsulées dans un processeur (Un nœud logiciel). Les quatre éléments : tâche, nœud logiciel, IP et nœud matériel, sont définis dans le paragraphe suivant.

2.1.1 Tâche

Par définition, une tâche est une unité logicielle qui s'exécute d'une façon séquentielle. Chaque tâche communique avec d'autres unités logicielles s'exécutant sur le même processeur.

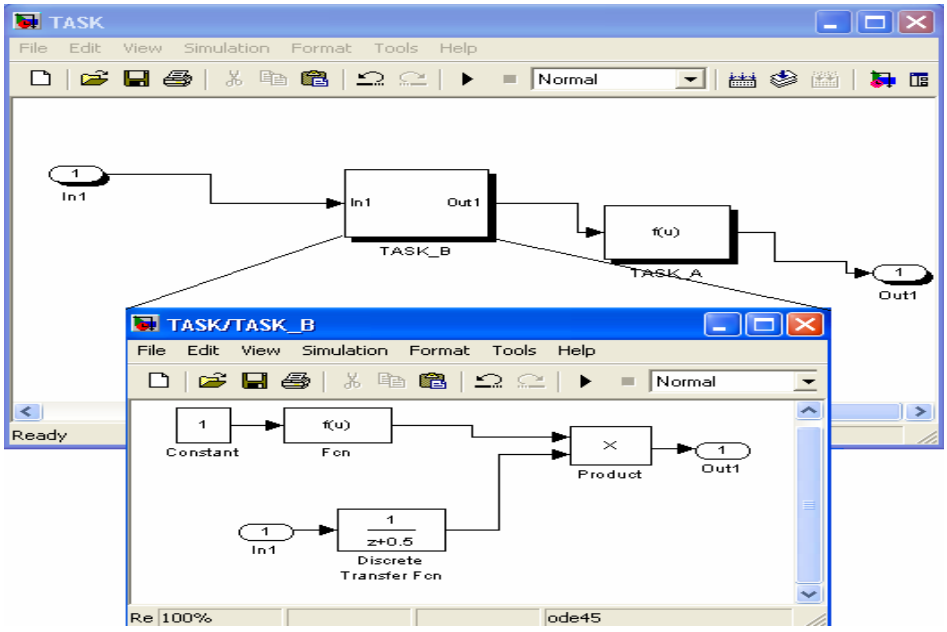


Figure 4.2: Représentation des tâches dans Simulink au niveau transactionnel

La Figure 4.2 représente deux tâches à l'intérieur d'un processeur. En effet, nous pouvons remarquer deux genres de tâches. La première tâche simple (TASK_A) est formée à partir d'un bloc prédéfini de la librairie. Par contre, la deuxième est hiérarchique (TASK_B) qui englobe plusieurs blocs de la librairie.

2.1.2 Nœud logiciel

Un nœud logiciel est un sous système qui contient plusieurs tâches qui se communiquent via des interconnections.

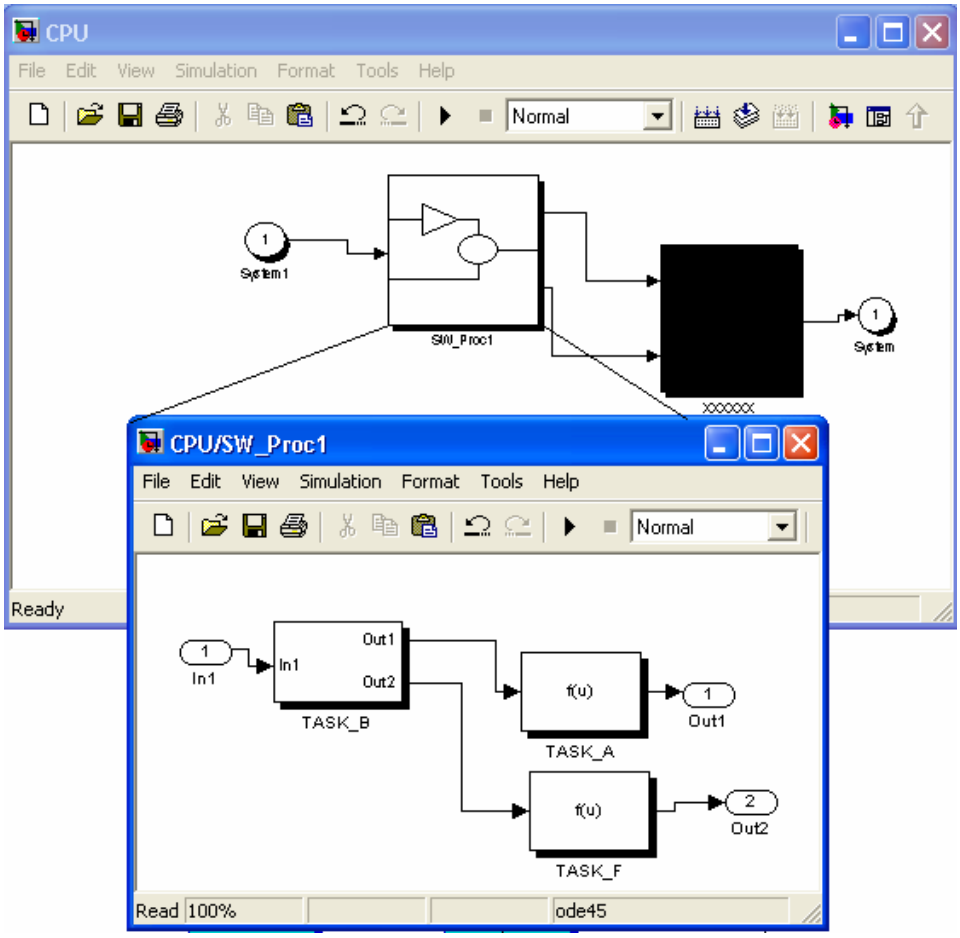


Figure 4.3: Représentation d'un noeud logiciel qui enveloppe trois tâches

Dans la représentation transactionnelle de Simulink, le nœud logiciel est un sous système hiérarchique de la librairie qui englobe des tâches dont le concept est déjà définit au paragraphe précédent. Ce sous système est caractérisé par le préfixe 'SW_' qui précède son nom. La Figure 4.3 représente un nœud logiciel transactionnel dans Simulink contenant trois tâches à l'intérieur ('TASK_A', 'TASK_B', 'TASK_F'). La communication du nœud logiciel avec le système se fait via des ports d'entrée et de sortie ('In1', 'Out1', 'Out2'). Le nom de ce processeur est 'SW_Proc1'. Le préfixe 'SW_' est indispensable pour identifier que ce sous système est un nœud logiciel.

2.1.3 IP matériel

D'une façon homologue à la tâche logicielle, l'IP matériel peut être formé par un bloc individuel prédéfini et également par plusieurs blocs de la librairie de Simulink. Cependant, le nom de ce sous système hiérarchique est précédé du préfixe 'IP_'.

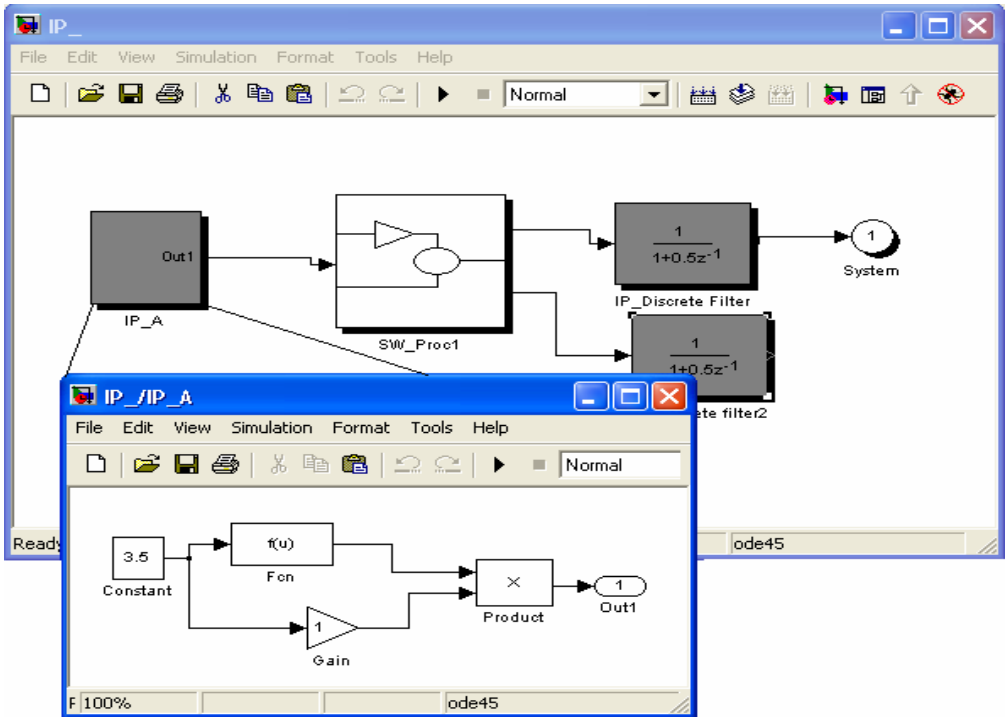


Figure 4.4: Représentation de diverses formes d'IP dans Simulink

La Figure 4.4 représente une architecture contenant deux types d'IPs définis dans Simulink. Un IP simple de la librairie formé d'un bloc de filtre discret et un autre hiérarchique contenant plusieurs blocs ('IP_A').

2.1.4 Nœud matériel

Le nœud matériel est un sous système qui englobe plusieurs IPs. Ces derniers peuvent être simples ou bien hiérarchiques (Paragraphe précédent). Le nom du nœud matériel doit être précédé du préfixe 'HW_'.

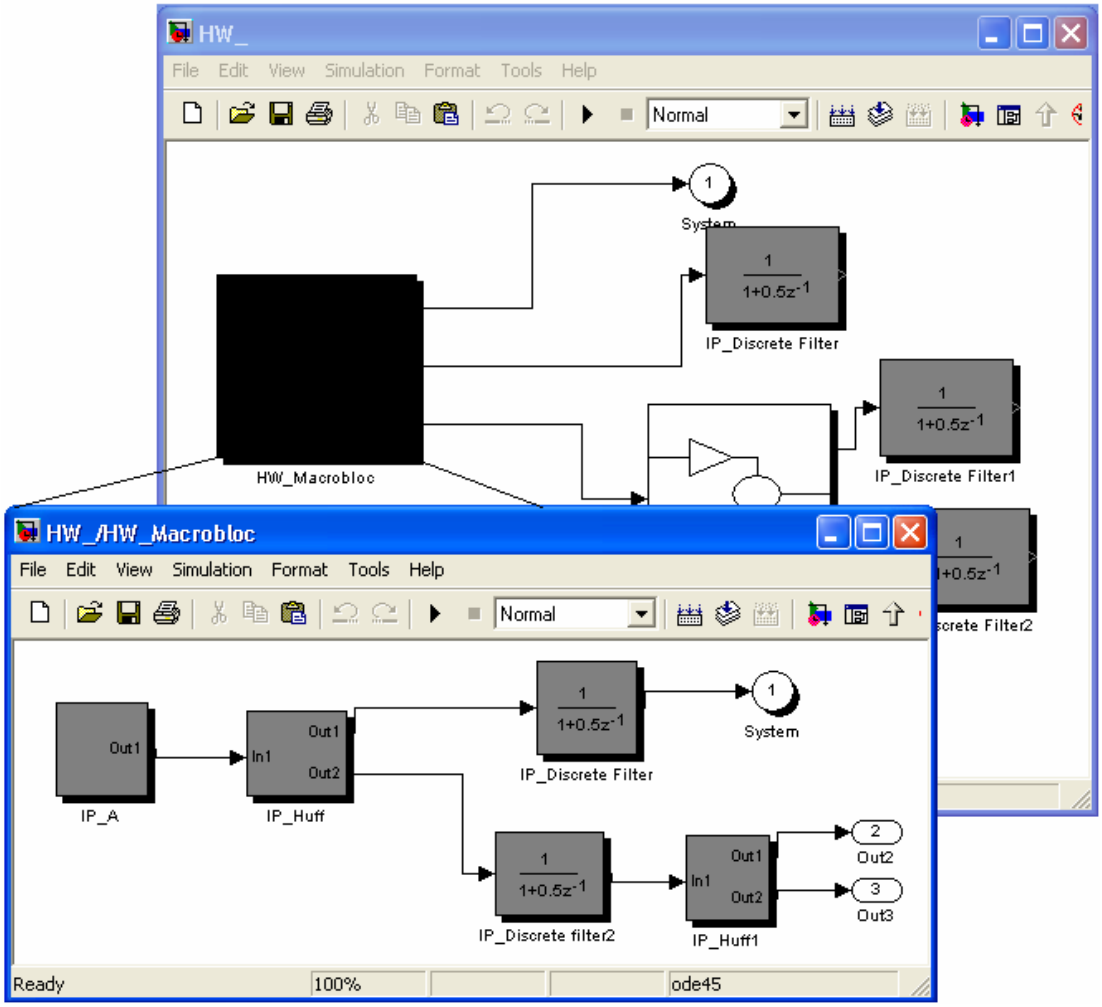


Figure 4.5: Représentation du nœud matériel dans un système transactionnel en Simulink

La Figure 4.5 illustre l'intégration d'un nœud matériel au sein d'un système en Simulink au niveau transactionnel. Le nœud matériel nommé 'HW_Macrobloc' contient cinq IPs.

2.2 Les composants de communication

Les composants de communication assurent le transfert des données entre les différents composants de calcul. Dans le modèle transactionnel proposé, ils sont formés par des blocs de la librairie de Simulink.

2.2.1 Topologie et protocoles de communication

La modélisation, que nous présentons au niveau transactionnel dans Simulink, offre la possibilité de modéliser la communication suivant plusieurs topologies, et selon plusieurs types de protocoles.

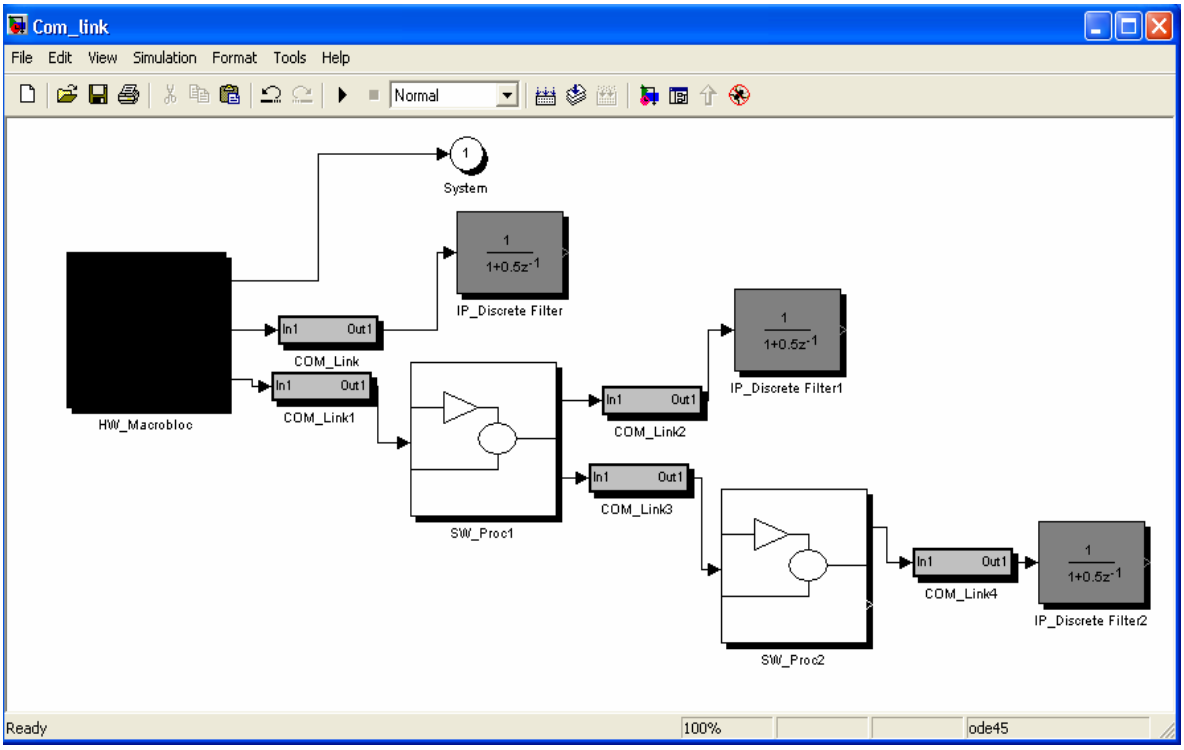


Figure 4.6: La communication point à point entre les différents sous systèmes

L'implémentation du comportement des blocs de communication n'est pas nécessaire pour effectuer la génération de l'architecture virtuelle. Pourtant, les protocoles de communication implémentés permettent la validation de la communication et de l'interaction entre les différents composants de l'architecture. La communication à ce niveau est effectuée à travers un sous système hiérarchique de la librairie qui présente le média entre les éléments de calcul. L'implémentation de son comportement peut être un FIFO, une mémoire partagée,….etc. Ce sous système de communication est caractérisé par son nom qui sera précédé du préfixe 'Com_'. La topologie de communication est flexible, elle peut être point à point, un bus partagé ou un réseau de communication. La Figure 4.6 illustre l'utilisation des sous système de communications suivant une topologie point à point. Néanmoins, la Figure 4.7 représente la communication suivant une topologie multipoint.

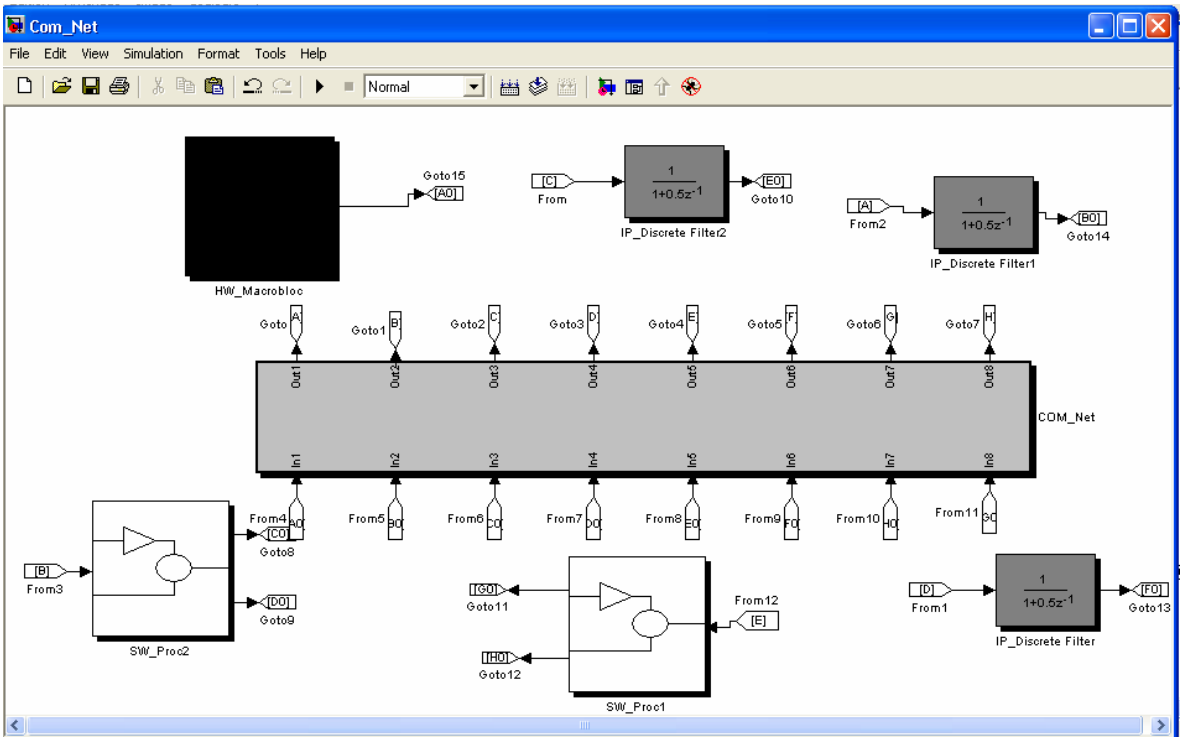


Figure 4.7: La communication multipoints entre les différents sous systèmes

2.2.2 Interface de communication

L'interface de communication assure le transfert des données entre un composant de calcul et le reste du système. Elle est formée en principe par des blocs de type 'inport' et 'outport' de la librairie. Cependant, elle peut être composée par des blocs ayant une certaine fonctionnalité. Par conséquence, le bloc interface assure la communication, convertit les signaux et adapte les différents composants de l'architecture. Simulink ne fournit pas des ports à double direction. Il y a uniquement des ports unidirectionnels d'entrée ou bien de sortie. Néanmoins, ces ports sont implicites dans le cas des blocs simples, et explicites dans le cas des sous systèmes.

3. Génération de la macro architecture (GMA)

La continuité entre le modèle intermédiaire en Simulink et l'architecture virtuelle en Colif est établie par l'intermédiaire de l'outil de génération de la macro architecture (GMA). L'entrée de cet outil est la description transactionnelle du système sous Simulink. Ce modèle est décrit sous forme de blocs interconnectés entre eux via des vecteurs de données et regroupés dans des sous systèmes (cf. paragraphe 2). La Figure 4.8 illustre les différentes étapes qui permettent de générer l'architecture virtuelle en Colif. En premier temps, l'outil génère un arbre intermédiaire à partir d'un fichier « .mdl ». Ce même arbre alimente le générateur de Colif et le générateur des paramètres. Les paramètres ainsi générés seront spécifiés en ce qui convient l'architecture. Ils seront utilisés pour annoter et raffiner le Colif. Les différentes phases de génération sont détaillées dans les paragraphes qui suivent.

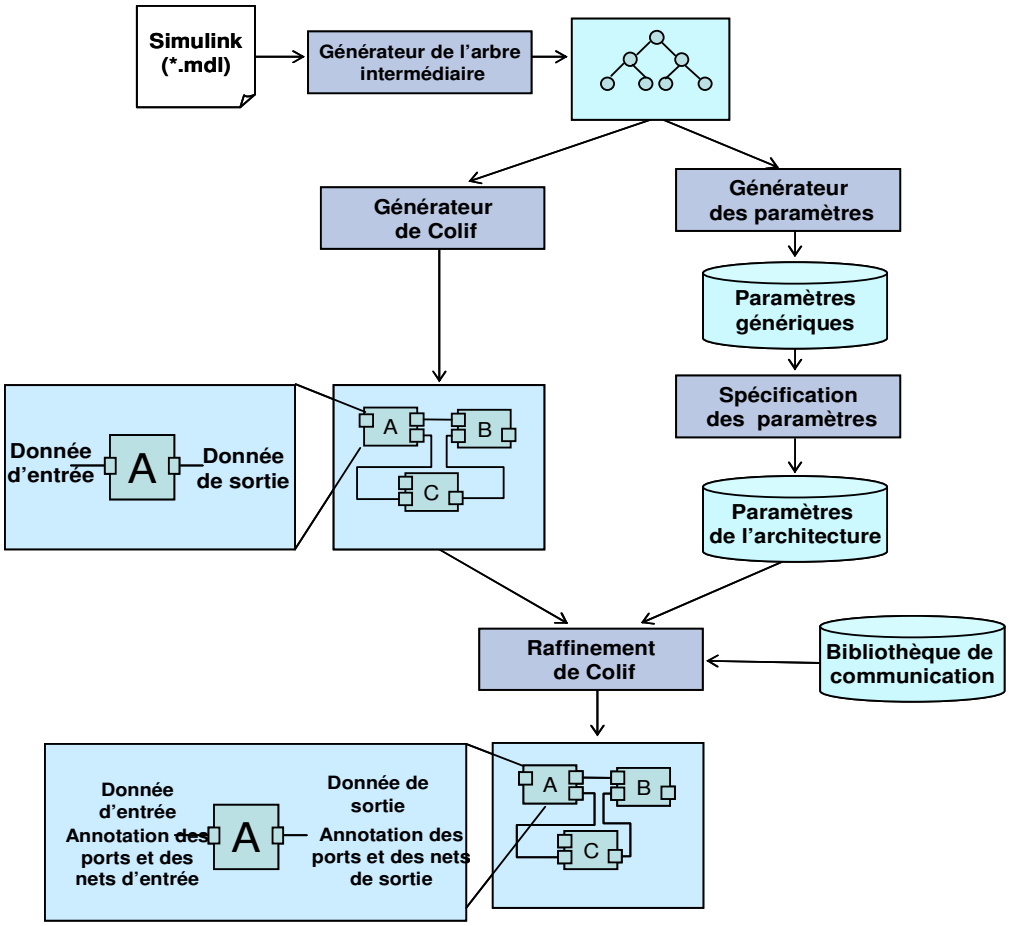


Figure 4.8: Les étapes de génération de l'architecture virtuelle

3.1 La description du fichier « .mdl » de Simulink

Simulink est un outil de conception et de simulation graphique de haut niveau. Il présente une énorme bibliothèque de composants pour différents champs d'applications. Derrière cette interface graphique interactive, il y a un fichier ASCII qui représente le système. C'est un fichier systématique (Figure 4.9) qui contient des mots clés, des paramètres et des valeurs qui décrivent le modèle. La partie principale du fichier est la partie "system" où les blocs et les connections (lignes et branches) sont décrites. Nous trouvons deux genres de blocs dans la bibliothèque de Simulink, des blocs simples et des blocs hiérarchiques. Les blocs simples sont définis par plusieurs paramètres dans le fichier ASCII. Nous nous sommes intéressés à "BlockType", "name", "ports" ou "port".

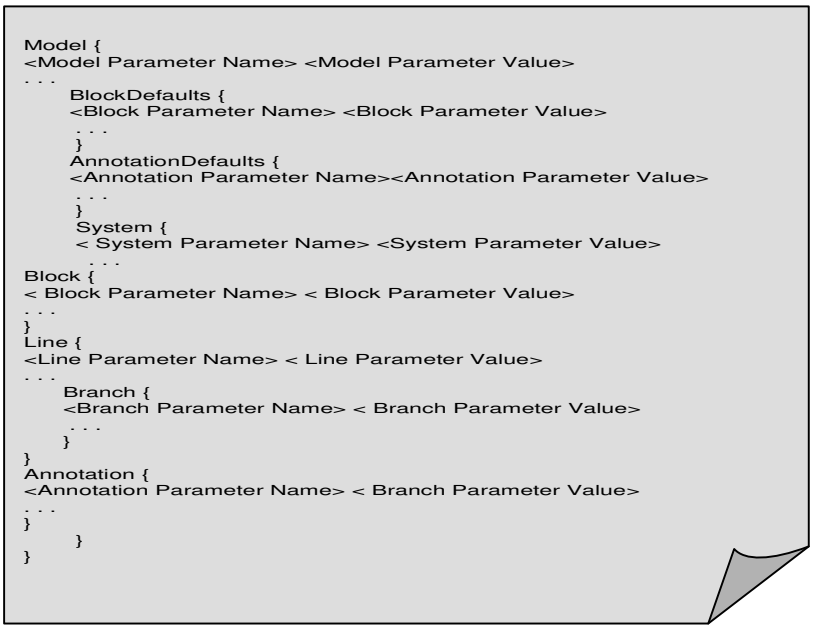


Figure 4.9: Le fichier ASCII générique de Simulink

Les blocs hiérarchiques ("sous-Systèmes") peuvent contenir plusieurs blocs, simples et hiérarchiques. Les ports d'entrée et de sortie du "sous-Systèmes" sont définis par l'intermédiaire de deux types de blocs fictifs "inport" et "outport". La section de "ligne" est formée à son tour par les paramètres "SrcBlock" et "DstBlock", leurs valeurs sont le nom du bloc de la source et de la destination. "SrcPort" et "DstPort" indiquent le port auquel la ligne est reliée. Enfin la section "branche" dérive de la section "ligne", où nous trouvons seulement les paramètres "DstBlock" et "DstPort" puisque la branche emploie le même bloc de source que la ligne.

3.2 Générateur de la forme intermédiaire (arbre intermédiaire)

Le générateur de la forme intermédiaire est développé en utilisant C++, lex et yacc qui sont sont utilisés pour la conception des compilateurs et la génération des translateurs de code. Chaque langage possédant une grammaire et des règles, peut être traité par lex. et yacc. Ceci permet une translation rapide, une modification facile, et une simple maintenance des programmes.

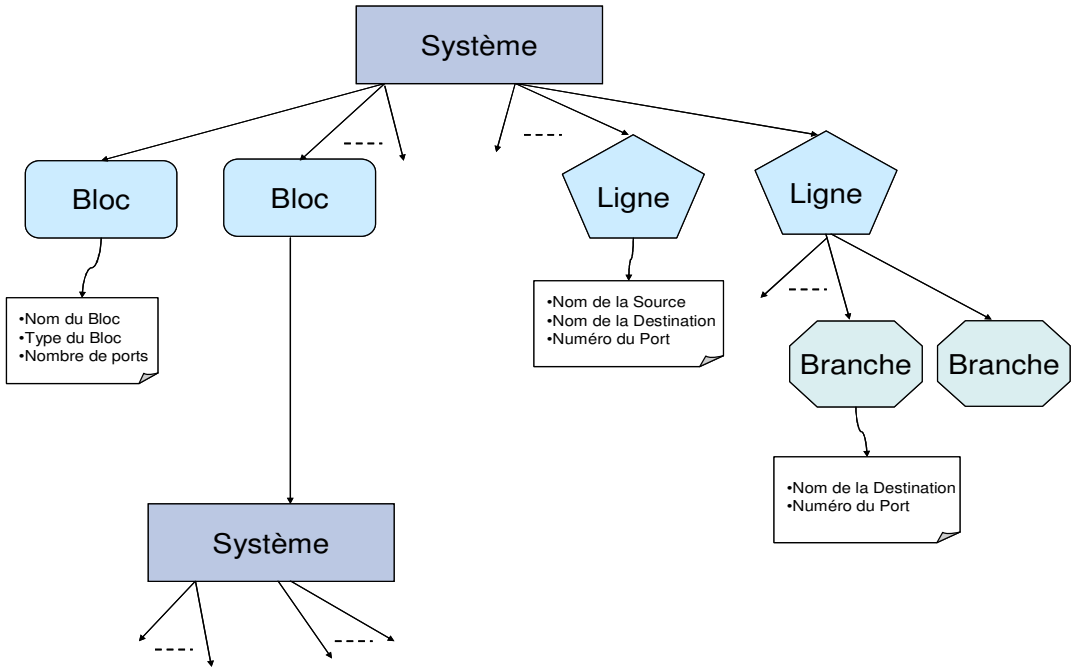


Figure 4.10: Structure de la base de donnée d'un système en Simulink

Le générateur admet comme entrée un fichier d'extension **«.mdl»**. Il traite le fichier suivant la grammaire de Simulink, en prenant en considération la hiérarchie et les connections. A la sortie du générateur, le fichier **« .mdl »** qui décrit le système Simulink sera stocké dans une base de données (Un arbre intermédiaire) formée à partir de plusieurs classes C++. Celle-ci contient toutes les informations propres aux blocs de Simulink et aux interconnections entre ces blocs. La structure de cette base de donnée présentée dans la Figure 4.10 est extensible et facile à modifier. Ceci permet d'ajouter facilement des informations additionnelles extraites de Simulink. L'exploitation de cette base de données mène à la génération de l'architecture virtuelle sous Colif dans l'étape qui suit.

3.3 Générateur des éléments de Colif

Le générateur de Colif est alimenté par la base de données produite par le générateur de l'arbre intermédiaire. La génération est faite en adaptant les concepts de

Simulink aux concepts de Colif (Figure 4.11). La génération de Colif [Ces 01] consiste à créer le système à partir de la génération de trois éléments de base qui sont le composant, l'interface et l'interconnexion.

3.3.1 Génération des composants

Un composant ou un module dans Colif peut être une tâche logicielle, un module virtuel logiciel, un IP matériel ou un module virtuel matériel. Dans Simulink, c'est le préfixe du nom des blocs hiérarchiques qui détermine le type du composant. En effet, le générateur de Colif parcourt et analyse l'arbre intermédiaire pour générer les composants de Colif. Il procède comme suit :

Il vérifie le cas où le bloc a un descendant 'système' (donc c'est un sous système), il détecte son nom et retire le préfixe. Si ce dernier est 'IP_' ou 'TASK_' l'outil génère directement un module simple sous Colif sans parcourir les descendants de l'arbre. Mais, si le préfixe est 'SW_' ou 'HW_' il génère respectivement un module virtuel logiciel ou matériel et parcourt l'arborescence. D'autre part, les blocs simples (n'ont pas de descendants) qui se trouvent à l'intérieur d'un sous système logiciel ('SW_') sont considérés comme des tâches. Ceux qui se trouvent dans un sous système matériel ('HW_') ou bien qui sont non encapsulés, sont supposés être des IPs matériels. Pour créer un module Colif, il faut instancier des objets de type ColifModuleContent, et ColifModuleEntity.

a) L'entité de chaque module comporte :

- Une référence vers un objet définissant le type du module. Il faut donc instancier un objet de type ColifModuleType. Ce dernier contient une référence vers un objet dont le type est l'un des types prédéfinis par Colif. Un module peut être soit un processeur (ColifCpu), une tâche logicielle (ColifSoftware), un bloc matériel (ColifHardware), une boite noire (ColifBlackBox) ou un composé d'autres types de modules (ColifCompound) ;
- Une référence vers une liste de déclaration de port.

b) Le contenu de chaque module comporte :

- Une référence vers une liste de déclaration de modules (les sous modules). Cette liste est vide lorsqu'il s'agit d'un module feuille dans la hiérarchie. Par exemple un module de type processeur contient un ou plusieurs sous modules de type logiciel (tâches) qui sont des feuilles dans la hiérarchie.
- Une référence vers une liste de déclaration de net. Cette liste est vide pour les modules feuilles dans la hiérarchie. Dans le cas contraire, elle représente l'ensemble des canaux qui connectent les sous modules qui correspondent au module courant.
- Une référence vers une liste d'objets de type ColifModuleBehaviour qui représentent le comportement interne du module. Cette liste est vide pour les modules de type processeur ou de type composé. Un objet de type ColifModuleBehaviour contient une référence vers une liste d'objets de type ColifSource qui encapsulent les liens vers les fichiers de comportement du module et les noms des langages de programmation utilisés.

3.3.2 Génération des interfaces

L'interface permet d'établir le lien entre le composant et l'interconnexion. Dans Colif l'interface est formée par un port simple qui correspond à un composant élémentaire (tâche ou IP) ou bien par un port complexe, composé d'un port interne et d'un port externe. Le port complexe est attaché à un module virtuel matériel\logiciel. D'autre part, Simulink possède des interfaces implicites correspondantes aux blocs élémentaires (non hiérarchiques) et des ports explicites correspondants aux sous systèmes hiérarchiques. Les ports implicites de Simulink ne se configurent pas dans la grammaire, ils sont détectés en parcourant l'arbre à travers la source et la destination de chaque ligne d'interconnections. Par suite à chaque port d'une tâche ou d'un IP matériel inclu dans un sous système de Simulink l'outil génère un port simple sous Colif.

Quant aux sous systèmes hiérarchiques représentant les processeurs et les blocs matériels, leurs ports sont explicites. Ces ports sont des blocs élémentaires de type 'Inport' et 'Outport' d'entrée et de sortie respectivement. Leurs interfaces générées en Colif à partir

de ces ports explicites sont des ports composés dans le cas où ils sont attachés à un nœud logiciel ou à un nœud matériel. Ils sont des ports simples dans les autres situations.

Cependant, pour créer un port Colif, il faut instancier des objets de type ColiPortContent, et ColifPortEntity.

a) L'entité de chaque port comporte :

-Une référence vers un objet qui définit le type de port. Il faut donc instancier un objet de type ColifPortType. Ce dernier spécifie le type et la direction du port : réception de données (IN), émission de données (OUT), réception/émission de données (INOUT). La direction est inconnue (UNKNOWN) pour les ports de type SAP (Service Access Port) qui ne sont reliés à aucun canal de communication. L'objet de type ColifPortType contient aussi une référence vers une liste de définition de paramètres.

La définition d'un paramètre consiste à créer un objet de type ColifParamDefinition qui encapsule le nom du paramètre et le type de ses valeurs.

b) Le contenu de chaque port comporte :

- deux références vers deux listes de déclaration de ports (les sous ports). Dans le cas de ports virtuels, ces listes contiennent l'ensemble des ports internes et externes spécifiques au port virtuel.

3.3.3 Génération des interconnections

Les interconnections sous Colif appelées 'nets' permettent le passage des données aux différents composants en établissant le lien entre les interfaces des modules. Toutefois, nous distinguons deux types d'interconnections : les nets simples et les nets virtuels. Les nets simples relient les interfaces des modules simples entre eux ou bien avec les ports internes des modules virtuels. Par contre, les nets virtuels relient les ports composés (interne\externe) des modules virtuels entre eux. Dans Simulink TLM, les interconnections se font via des lignes (Trait simple unidirectionnel) et des sous systèmes de communications. Ces derniers peuvent implémenter différents protocoles de communications. Les nets générés par l'outil dans Colif sont simples dans le cas où

l'interconnexion dans Simulink est intra processeur et intra Macro bloc matériel. Cependant Ils sont virtuels dans le cas où l'interconnexion Simulink est inter nœud logiciel, nœud matériel. Pour créer un net Colif, il faut instancier des objets de type ColiNetContent, et ColifNetEntity.

a) L'entité de chaque canal (net) comporte :

-Une référence vers un objet qui définit le type de net. Il faut donc instancier un objet de type ColifNetType. Ce dernier contient une référence vers une liste de définition de paramètres.

b) Le contenu de chaque canal (net) comporte :

- Une référence vers une liste de déclaration de nets (les sous nets). Cette liste contient l'ensemble de sous canaux spécifiques à un canal virtuel donné.

- Une référence vers une liste d'objets de type ColifNetBehaviour qui spécifie le nom du protocole utilisé par un canal donné.

Après la définition des différents modules, ports et nets, nous déclarons les modules, les ports et les canaux présents dans le système. En Colif, ceci revient à créer respectivement des objets de type ColifModuleDecl, ColifPortDecl et ColifNetDecl. L'ensemble de ces objets représente des références vers les définitions de modules, ports et nets qui seront instanciés. En plus de ces déclarations, il faut instancier aussi un objet de type ColifRoot. Cet objet représente le module racine de la hiérarchie. Le format Colif ne supporte qu'un seul et unique objet de ce type (ColifRoot). Enfin, nous instancions les différents objets déclarés. En Colif, ceci crée un arbre d'instances composé par des objets de type ColifModuleInstance, ColifPortInstance et ColifNetInstance. Le parcours de cet arbre permet un accès aux données de conception à travers une couche API offerte par Colif. La Figure 4.11 illustre la génération des trois éléments de base (le module, le port et le net) de l'architecture virtuelle en Colif à partir des éléments similaires qui définissent l'architecture transactionnelle en Simulink.

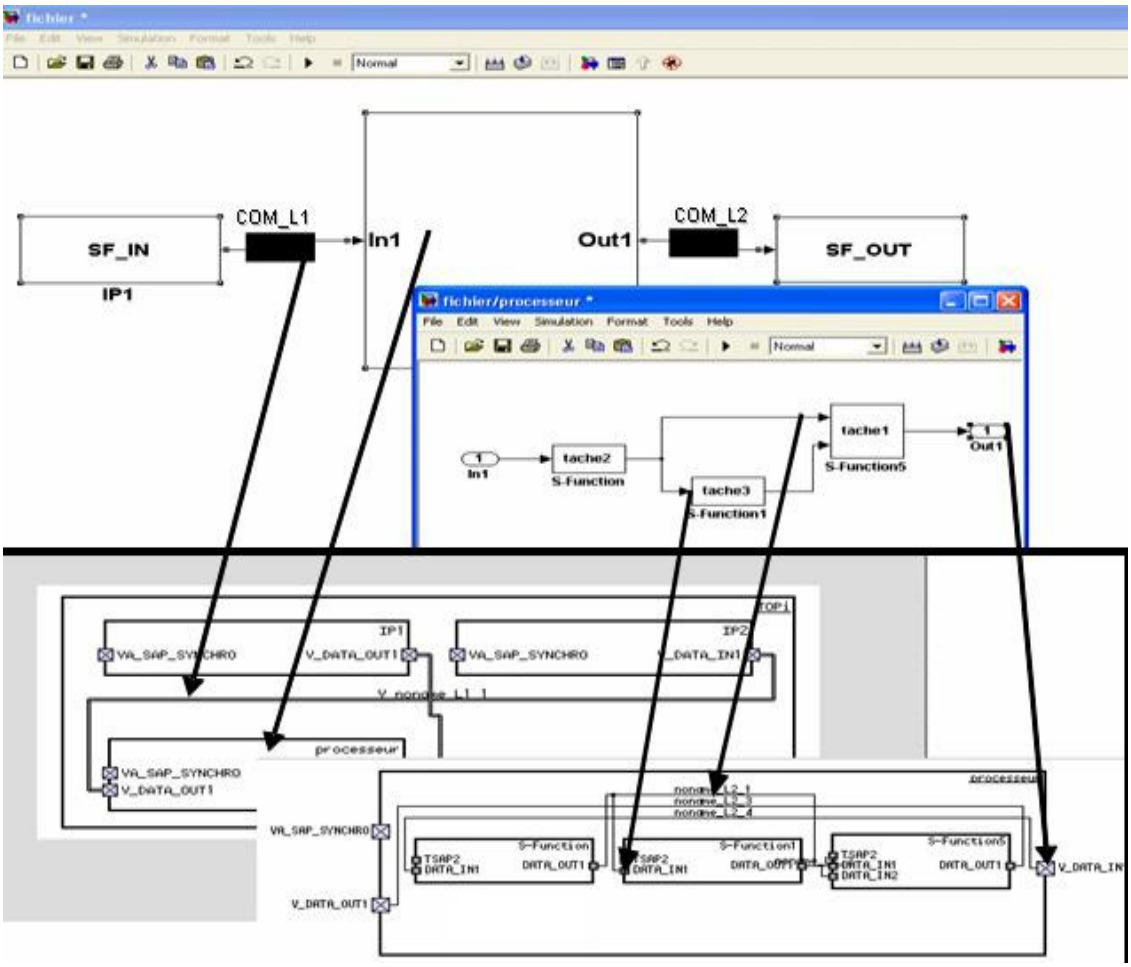


Figure 4.11: Génération de Colif à partir de Simulink

3.4 Génération et spécification des paramètres de l'architecture virtuelle

La sémantique sous Colif est représentée sous forme de paramètres attachés aux modules, ports et net. Par conséquent, chaque module/Port/net doit avoir un certain nombre de paramètres contenant les informations de conception nécessaires. Ces paramètres sont indispensables pour réaliser la cosimulation au niveau macro architecture et au niveau micro architecture et pour synthétiser et raffiner l'architecture dans le flot de conception.

Sous Colif, chaque objet de type ColifModuleInstance, ColifPortInstance et ColifNetInstance contient une référence vers une liste d'objets de type ColifParameter :

ce type d'objet encapsule le nom et la valeur d'un paramètre. Les tableaux (Tableau 4.2**,** Tableau 4.3**,** Tableau 4.4**,** Tableau 4.5) résument l'ensemble des paramètres qui doivent être définis respectivement dans un module, dans un port (interne\externe), et dans un net. Dans ces tableaux nous pouvons voir en plus, la sémantique de chaque paramètre et les outils pour lesquels chaque paramètre est spécifique. Les outils HW, SW et co-Sim représentent respectivement l'outil de génération d'interfaces matérielles, l'outil de génération d'interfaces logicielles et l'outil de génération d'adaptateurs fonctionnels pour la co-simulation.

Module		
Paramètres	**Sémantique**	**Outil**
Adress_INT	Adresse physique du gestionnaire de requêtes d'interruption	HW/SW
CPU	Identifiant d'une architecture local d'un processeur	HW/SW
DATA_INT_WIDTH	Largeur du bus interne au Coprocesseur de communication	HW

Tableau 4.2: **Sémantiques, et outils utilisés par les paramètres du module**

Port interne		
Paramètres	**Sémantique**	**Outil**
ADRESS_DATA	Section pour l'accès au canal	HW/SW
ADRESS_STATE	Section pour l'accès au registre d'état du canal	HW/SW
CHAN_NUM	Identifiant de canal pour la vectorisation des requêtes d'interruption	HW/SW
MASK_GET	Position du drapeau de lecture	HW/SW
MASK_PUT	Position du drapeau d'écriture	HW/SW
SoftPortType	Identifiant du pilote	HW/SW
DATA_TYPE	Type C des données	HW/SW
IT_LEVEL	Priorité d'irq	HW/SW
DATA_BIT_WIDTH	Longueur native des données en interne au module	HW
Poliferation	Réplication de module, port ou canal pour une mise à l'échelle de l'architecture	HW
SystemCType	Type, protocole, niveau d'abstraction d'un port ou d'un canal	Co-Sim
SystemCDataType	Type de données échangées par le canal connecté au port	Co-Sim

Tableau 4.3: Sémantiques, et outils utilisés par les paramètres du port interne

Port externe		
Paramètres	**Sémantique**	**Outil**
DATA_BIT_WIDTH	Longueur du média externe inter modules	HW
CHAN_PRIO	Priorités relatives des canaux pour les requêtes d'interruptions	HW
HardPortType	Identifiant du protocole RTL	HW
POOL SIZE	Taille en nombre de mots du buffer interne au adaptateur de canal	HW
SHB_MASK	Masque pour la translation des adresses	HW
SHB_OFFSET	Section de mémoire assignée à un port esclave	HW
SHB_ADRESS	Section de mémoire assignée à un port esclave	HW
SHB_PRIO	Priorité statistique ou rapport cyclique pour l'accès à un bus partagé	HW
Poliferation	Réplication de module, port ou canal pour une mise à l'échelle de l'architecture	HW
SystemCType	Type, protocole, niveau d'abstraction d'un port ou d'un canal	Co-Sim
SystemCDataType	Type de données échangées par le canal connecté au port	Co-Sim

Tableau 4.4: **Sémantiques, et outils utilisés par les paramètres du port externe**

Canal		
Paramètres	**Sémantique**	**Outil**
Poliferation	Réplication de module, port ou canal pour une mise à l'échelle de l'architecture	HW
SystemCType	Type, protocole, niveau d'abstraction d'un port ou d'un canal	Co-Sim
SystemCDataType	Type de données échangées par le canal connecté au port	Co-Sim

Tableau 4.5 Sémantiques, et outils utilisés par les paramètres du canal

Nous avons classifié ces paramètres en deux catégories pour pouvoir les gérer et les joindre facilement à l'architecture virtuelle. La première catégorie correspond aux protocoles de communication, dans laquelle les valeurs de ces paramètres caractérisent et dépendent du protocole adopté. Cet ensemble de fichiers forme la bibliothèque des paramètres de protocoles de communication. La deuxième catégorie, dépend de l'architecture et doit être personnalisée par l'utilisateur.

a) Bibliothèque des paramètres de protocoles de communication

La bibliothèque de ces paramètres contient des fichiers qui comportent tous les paramètres des ports (interne\externe) et des nets qui correspondent à chaque type de communication utilisé. Nous trouvons (SHM.txt, PIPE.txt, EVENT.txt….). Chaque fichier de communication a un format bien précis. La Figure 4.12 représente un fichier de paramètres propres au protocole de communication 'EVENT'. Ces paramètres correspondent à des 'Nets' et à des 'Ports'. La valeur du paramètre 'SystemCType ' dépend de la direction du signal (entrant ou sortant).

```
Fichier protocole <Event.txt>
<NET>
IT_LEVEL: T
SystemCType: va_ch_mac_event
<Port>
SoftService: Kernel/signal/wakeup
SystemCType (Direction:: IN, OUT): (va_in_mac_event, va_out_mac_event)
```

Figure 4.12:Les paramètres des protocoles de communications

b) Fichiers de paramètres spécifiques à l'architecture

Après l'analyse de l'arbre intermédiaire, le générateur de paramètres produit à la sortie des fichiers décrivant le système, les nœuds matériels et les nœuds logiciels. Ces fichiers offrent à l'utilisateur la possibilité d'ajouter une sémantique aux éléments de l'architecture en annotant la représentation Colif par des paramètres architecturaux. L'outil génère un fichier pour le système entier décrivant les paramètres des modules virtuels et de leurs interconnections. Il génère également un fichier pour chaque nœud matériel\logiciel décrivant les paramètres des modules simples encapsulés et de leurs interconnections.

```
Fichier <Module virtuel.txt>
<Nom_du_module1>
TaskPriority: xxx
<Nom_du_module2>
TaskPriority: xxx
……..
<Name_NET1 : [Module1/Port ; Module2/Port ; ……] >
                                    Protocole: xxx; //(SHM, PIPE)
SystemCDataType: xxx |SystemCDataBitWidth: xxx |IT_NUMBER: xxx ;
<Module1/Portxx>
HardPortType: xxx |POOL_SIZE: xxx |ADDRESS_DATA: xxx |ADRESS_STATE: xxx;
```

Figure 4.13: Les paramètres spécifiques à l'architecture

La Figure 4.13 représente le fichier générique 'template' généré à partir de l'arbre intermédiaire. Ce fichier de paramètres correspond à un nœud matériel\logiciel ou à l'architecture globale. Le concepteur intervient pour spécifier la valeur des paramètres ayant une valeur 'xxx'. Le but est d'emporter ces paramètres à la description Colif et aboutir à un prototype virtuel simulable et synthétisable.

3.5 Raffinement de Colif

Le raffineur de Colif analyse la description Colif, détecte chaque élément (module, net, port) et trouve les paramètres correspondants à chacun de ces éléments dans le fichier spécifique aux paramètres. L'outil retire les paramètres et leurs valeurs, et annote le modèle Colif. Ces paramètres apparaissent quand nous cliquons sur la représentation graphique de Colif. Il raffine également l'architecture en ajoutant une tâche "sandby" à l'intérieur de chaque nœud logiciel. La tâche "standby" est servie par le système d'exploitation, quand toutes les autres tâches sont interrompues. L'outil ajoute aussi un port « SAP » pour chaque module logiciel pour assurer la synchronisation des tâches au moment de la cosimulation.

4. Génération des tâches en SystemC à partir de Simulink

Les tâches au niveau transactionnel sous Simulink sont incluses dans un nœud logiciel représenté par un sous système ayant le préfixe 'SW_ ' dans son nom. Ces tâches sont modélisées sous Simulink de plusieurs façons (Figure 4.14).

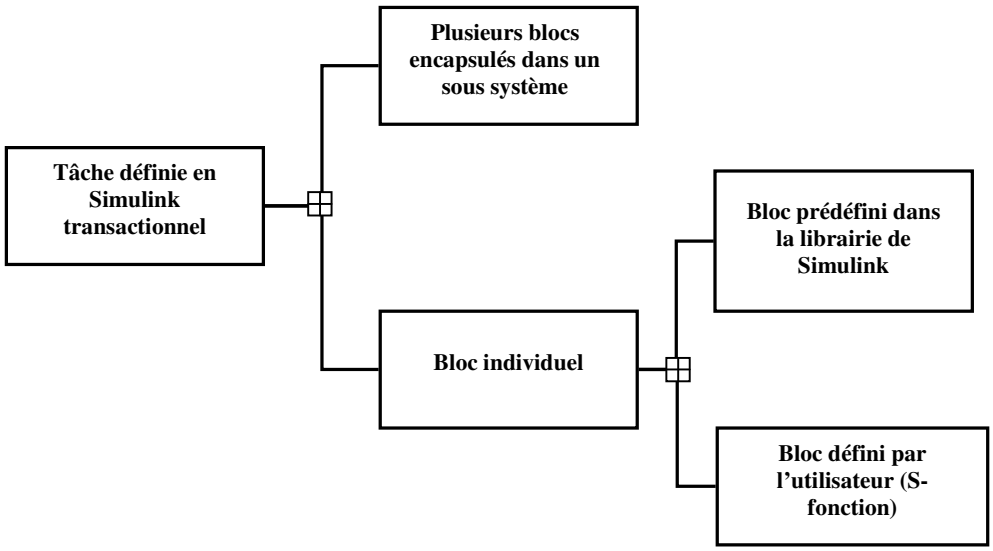


Figure 4.14: Les différents types de tâches en Simulink transactionnel

Elles peuvent être formées par un fusionnement de plusieurs blocs dans un sous système ayant le nom précédé du préfixe 'TASK_' ou bien elles sont formées par des blocs individuels. Ces derniers, à leur tour peuvent être des blocs prédéfinis de la bibliothèque ou bien des S-fonctions modélisées en langage C. Dans ce qui suit, la modélisation des tâches en SystemC sera expliquée avant de décrire les différentes façons admises pour transformer les tâches de Simulink transactionnel en tâches décrites en SystemC. Notons que l'automatisation de la génération des tâches SystemC à partir de Simulink sera aisée après l'explication menée dans ce paragraphe des différents modèles et cas confrontés.

4.1 Description des tâches en SystemC

Pour la modélisation des tâches (modules), le SystemC [Sys], [Sys 05], utilise la notion de « SC_MODULE ». Un module peut être hiérarchique contenant d'autres modules, ou élémentaire contenant un comportement actif ou passif, ceci grâce aux différentes primitives de modélisation de comportement (« SC_METHOD », « SC_THREAD » ou « SC_CTHREAD »). Dans notre cas nous utilisons les modules élémentaires de type « SC_CTHREAD » qui présentent des avantages par rapport à SC_METHOD; Il est possible de suspendre la tâche « SC_THREAD » au moyen de la

fonction "wait ( ) ". Par contre la fonction "wait ( )" utilisée au sein de « SC_METHOD » génère une erreur pendant l'exécution.

```
SC_MODULE (SF_SYNCRO)
{
  va_in_mac_pipe <long int> DATA_IN1;

  va_out_mac_pipe <long int> DATA_OUT1;
  va_synchro                    TSAP2;
  Void SF_SYNCRO_beh ();
  SC_CTOR (SF_SYNCRO)
   {
            SC_CTHREAD (SF_SYNCRO_beh, TSAP2.pos ());
   };
};
```

Figure 4.15: Exemple d'un fichier entête '.h' correspondant à une tâche SystemC

D'autre part, la communication est déterminée à travers une interface de communication. Cette dernière est décrite à travers un ensemble de ports qui peuvent être de différents types : des ports d'entrées, des ports de sorties et des ports d'entrées/sorties. SystemC fournit également un port spécifique pour la modélisation d'une horloge physique.

```
#include <stdio.h>
#include <stdlib.h>
#include "SF_SYNCRO.h"
void SF_SYNCRO::SF_SYNCRO_beh()
{
long int entree1;
long int sortie1;
            For (;;)
            {
            entree1=DATA_IN1.Get( );


            //calcul

            DATA_OUT1.Put(sortie1);
            };
}
```

Figure 4.16: Exemple d'un fichier '.cpp' correspondant à une tâche SystemC

La Figure 4.15 représente le fichier entête d'une tâche décrite en SystemC. L'interface de ce module est formée par un port d'entrée et un port de sortie de type 'long int'. La tâche possède un port de service 'SAP'qui permet la synchronisation des tâches dans la cosimulation. Cependant, la Figure 4.16 représente le fichier principal '.cpp'. Le calcul principal est effectué au corps de cette tâche. La communication de ce module avec le système se fait à travers les interfaces représentées par les ports d'entrées et de sorties 'DATA_IN1' et 'DATA_OUT1' au moyen des APIs définies dans la librairie.

4.2 Transformation des S-fonctions de Simulink en tâche SystemC

SystemC est utilisé par les outils de synthèse et de cosimulation dans le flot de conception des MPSoCs proposé. Le processus de conception débute toujours par la spécification de l'application. Celle ci est développée dans l'environnement Simulink en utilisant parmi d'autres, des blocs S-fonctions.

Les S-fonctions sont développées en langage C suivant des règles précises et à travers des méthodes décidées par le simulateur de Simulink. Une S-fonction est formée de quatre méthodes indispensables. Dans notre travail, un bloc S-fonction sera converti en un module en SystemC formé par un 'thread' sensible à un signal 'SAP'. Le fichier S-fonction.c sera transformé d'une façon directe en un fichier d'entête et un fichier d'implémentation en C++. La Figure 4.17 présente un exemple de transformation d'un bloc S-fonction en un module SystemC. Pour mieux comprendre la transformation d'une S-fonctions en une tâche SystemC, nous la décomposons en quatre parties.

Partie I : Dans cette partie, on définit les variables globales et on y inclut les fichiers d'entête '.h'.

- *S- fonction :* Les fichiers d'entête de la librairie de Simulink (Simstruct.h, …), les macros, les fichiers d'entête du code, et les variables globales sont définies.

- *SystemC :* Les fichiers d'entête de la librairie de SystemC, les macros, les fichiers d'entête du code et les variables globales sont définies.

Partie II : L'initialisation des variables et la définition des ports d'entrée et de sortie sont inclus dans cette partie.

- *S- fonction :* Cette partie est formée par la méthode mdlInitializeSizes (SimStruct *S) où des variables sont initialisées, et le nombre et la taille des ports d'entrée et de sortie sont définis.

- *SystemC :* Cette partie est répartie sur le fichier d'entête et le fichier d'implémentation de SystemC. Dans le premier, le type des ports est défini. Dans le deuxième les ports du module sont déclarés et initialisés. Le type du port dépend du type de communication servi par le port. (Mémoire partagée, Fifo, signal de synchronisation).

Partie III : Les APIs de communication et le calcul principal sont développés dans cette partie suivant une boucle qui se répète plusieurs fois.

- *S- fonction :* La méthode mdloutput (SimStruct *S) est utilisée dans cette partie. Le calcul principal du bloc est réalisé. Les données à transmettre sont affectées dans les ports au moyen de l'opérateur « = ». Ce dernier représente une primitive de communication.

- *SystemC :* La boucle for (; ;) dans le fichier d'implémentation renferme le calcul principal du module. Le code du calcul en langage C est similaire à celui de la S-fonction. La différence dans cette partie se manifeste au niveau des primitives de communication. Dans S-fonction, la lecture et l'écriture des données dans les ports se fait par l'affectation de l'opérateur **« = »**.

Dans SystemC il y a deux types de primitives de communications :

- Le **Get ( )** et le **Put ( )** pour la communication à travers un **« FIFO ».**
- l'opérateur **« = »** pour lire et écrire dans la mémoire partagée.

Partie IV : C'est la dernière partie qui s'exécute à la fin de la simulation.

- *S- fonction :* Cette partie est formée par la méthode mdlterminate (SimStruct *S).

- *SystemC :* Cette partie se trouve après la fin de la boucle for (;;) de la partie III et la fin du module.

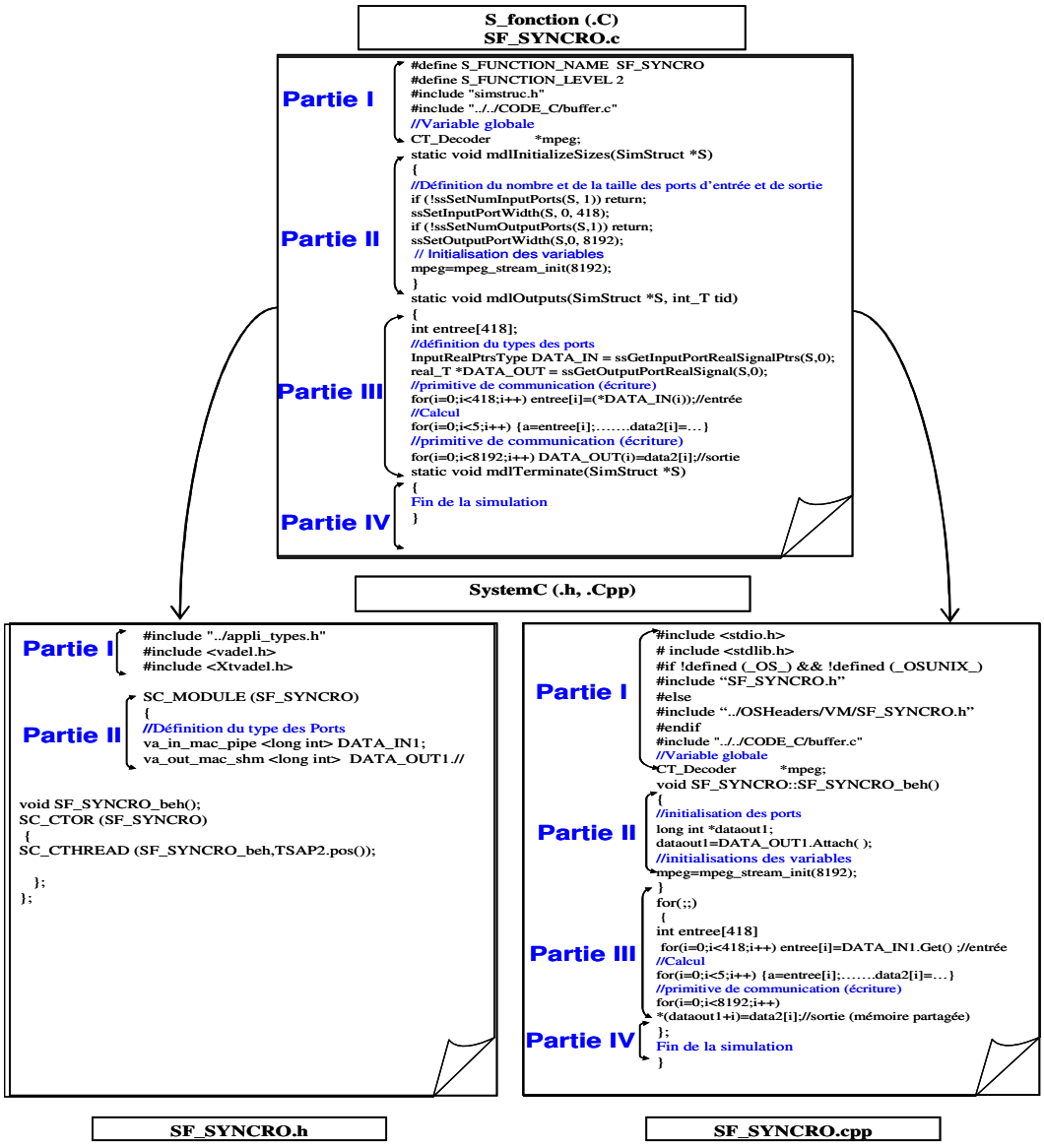


Figure 4.17: La transformation de la S_Fonction en SystemC

4.3 Création d'une tâche SystemC à partir d'un bloc prédéfini dans la librairie de Simulink

Dans le cas d'un bloc élémentaire de type différent du S-fonction, inclu dans un nœud logiciel (un sous système ayant le préfixe 'SW_'), la génération des tâches SystemC se fait à partir d'une librairie de fonctions décrivant le comportement de l'ensemble des blocs Simulink utilisés dans l'application.

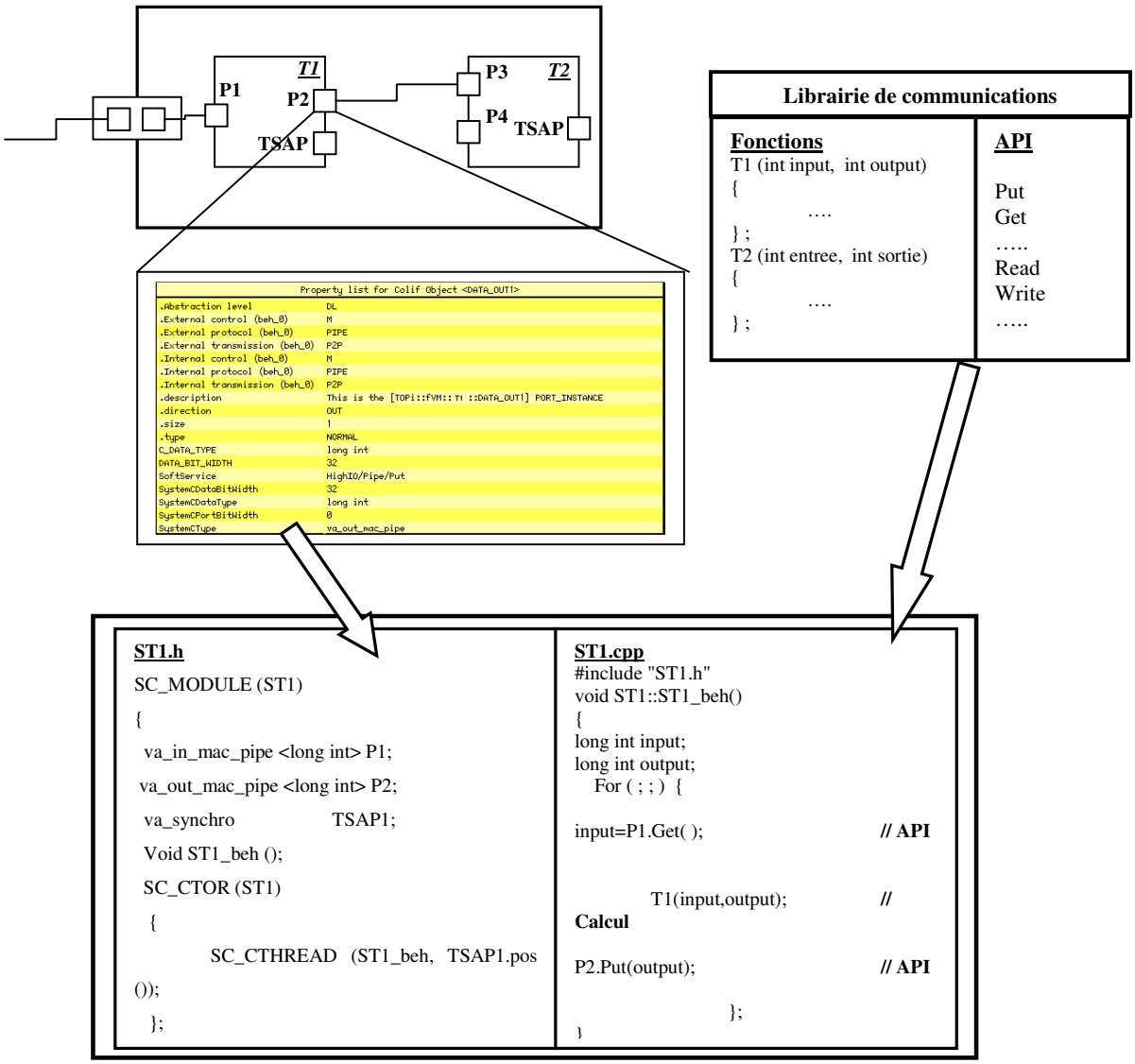


Figure 4.18 : La génération d'une tâche à partir d'un bloc élémentaire

Chaque fonction a le même nom que le bloc Simulink et le module correspondant en Colif. Cependant, la lecture et l'écriture des données se font via des APIs spécifiques à chaque protocole de communication. Ces APIs existent dans la librairie de communication. Le type du protocole de communication est identifié dans le 'Port' de chaque module en Colif raffiné.

La Figure 4.18 représente la génération d'une tâche en SystemC à partir d'un bloc individuel sous Simulink transactionnel, ce bloc étant transformé en un module paramétré sous Colif.

4.4 Fusion de plusieurs blocs Simulink en une tâche SystemC

Dans le cas où plusieurs blocs sont regroupés dans un sous système représentant une tâche dont le nom est précédé du préfixe 'TASK_', la génération de la tâche SystemC se fait en assemblant plusieurs fonctions de la librairie en une seule tâche SystemC. Les fonctions ont les mêmes noms des blocs. Ces fonctions s'échangent les données par l'intermédiaire de variables communes. La communication avec le système 'inter_Thread' se fait via des APIs générés suivant le protocole de communication défini sous Colif.

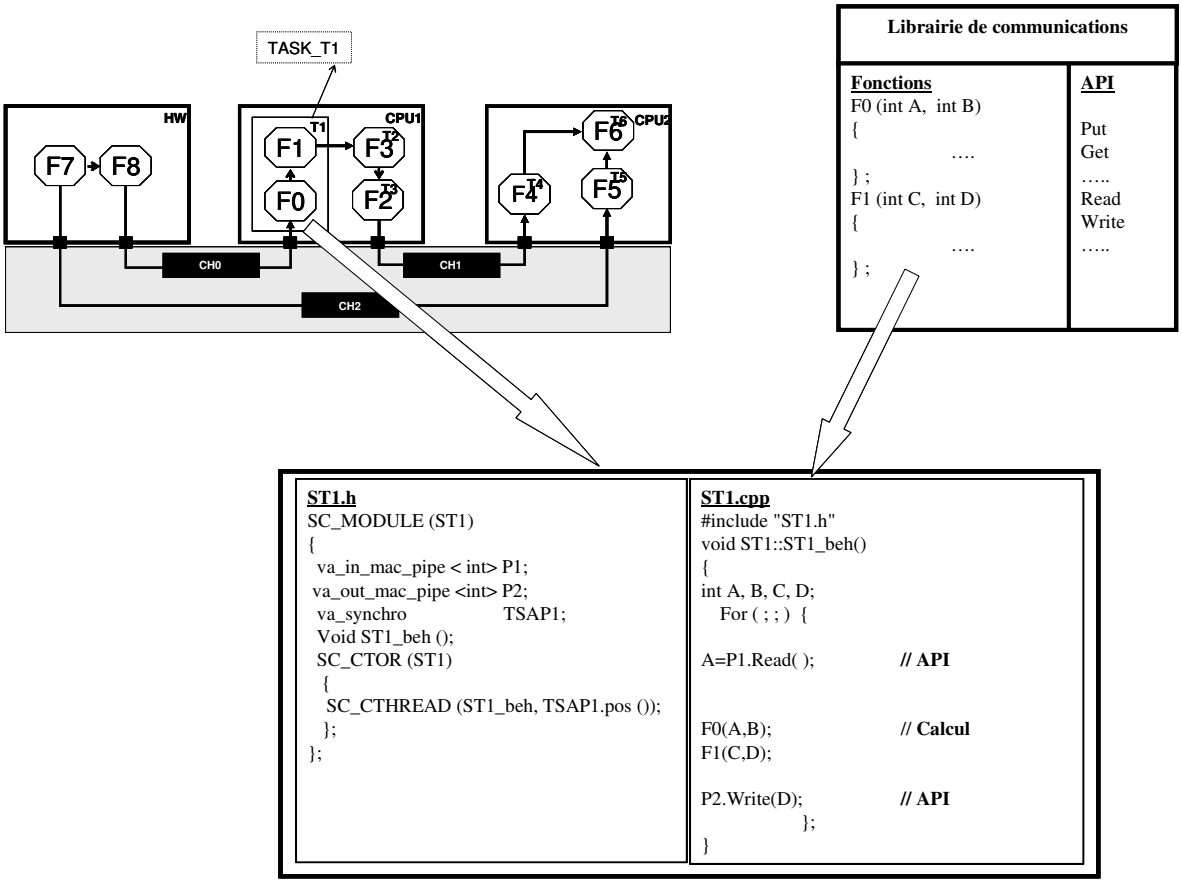


Figure 4.19: La génération d'une tâche à partir d'un ensemble de blocs sous Simulink

La Figure 4.19 illustre le fusionnement de plusieurs blocs sous Simulink transactionnel pour produire une tâche en SystemC. Les fonctions de la librairie F0 ( ), F1 ( ) portent les mêmes noms que les blocs F0, F1. La génération des APIs se fait en identifiant le type du protocole dans chaque port du module dans l'architecture virtuelle du Colif.

5. La synthèse de l'architecture (CA) à partir de l'architecture virtuelle

Une fois que nous avons l'architecture multiprocesseur virtuelle, les outils de synthèse interviennent pour raffiner cette architecture. Le rôle de ces outils est de mener l'architecture à un niveau plus bas où les détails de l'implémentation sont explicites. Le processus de raffinement consiste pour les nœuds logiciels à générer des interfaces matérielles et des interfaces logicielles. Cependant pour les nœuds matériels, ce processus consiste à raffiner les IPs matériels et corriger le comportement entre la description de haut niveau et celle de bas niveau. Le but ainsi visé est d'aboutir à une architecture synthétisable au niveau transfert de registre.

Notre architecture virtuelle est réalisée d'une façon adéquate pour qu'elle soit compatible aux outils de synthèse du groupe SLS qui sont ROSES et Macro cell builder. Ainsi en appliquant ces outils à notre architecture virtuelle, nous produisons une architecture multiprocesseur au niveau CA.

6. Conclusion

Dans ce chapitre nous avons présenté une solution pour combler l'immense fossé existant entre le modèle fonctionnel en Simulink et le modèle architectural en Colif. Cette solution a consisté à définir un modèle intermédiaire, transactionnel. Celui-ci a permit d'établir une continuité entre l'environnement Simulink et les outils de synthèse basés sur le langage intermédiaire Colif. Ce même modèle a aussi permis de combiner l'algorithme et l'architecture et de définir la plateforme d'implémentation à une étape de conception précoce.

L'outil 'Générateur de la macro architecture' (GMA), détaillé au cours de ce chapitre a guidé la transposition de la description fonctionnelle en une description architecturale. Cet outil était développé pour faire le lien entre l'environnement de conception et d'exploration d'algorithme (Simulink) et les outils de synthèse d'architecture (ROSES et Macro Cell builder).

Dans le chapitre suivant nous présentons une application multimédia et son implémentation à travers le flot de conception afin d'illustrer toutes les étapes de cette méthodologie et ainsi prouver son efficacité.

CHAPITRE 5

EXPERIMENTATION : LE DECODEUR MPEG LAYER III

Sommaire

1. Introduction

Dans les chapitres précédents, nous avons proposé un flot de conception des systèmes sur puce à partir d'une spécification au niveau système. Ce flot nous permet de procéder systématiquement à travers des niveaux d'abstraction progressifs pour arriver à une architecture au niveau micro architecture.

Dans ce chapitre nous appliquons la méthodologie de conception des MPSoCs sur un exemple réel, le décodeur MPEG layer III. Nous présentons tout d'abord l'algorithme standard de ce décodeur. Ensuite, nous le spécifions au niveau fonctionnel dans l'environnement Simulink. Après cette spécification, nous modélisons ce décodeur au niveau transactionnel TLM-Simulink. L'outil de génération de la macro architecture et les outils de synthèses d'architectures adoptés dans notre flot de conception seront appliqués sur ce modèle intermédiaire. Nous aboutissons à une architecture hétérogène au niveau transfert de registre.

Cette application nous a permis de valider notre approche de conception de haut niveau des MPSoCs et de prouver son efficacité.

2. Présentation du décodeur MP3

MPEG-1 layer-III, connu sous le nom MP3 [Iso 93], [Bra 00] est un format audio numérique à taux de compression élevé. Il permet de diminuer de presque 12 fois la taille du fichier audio. L'intérêt principal de ce format, est qu'il fournit un taux de compression significatif sans dégrader la qualité de l'écoute du son. Tandis qu'auparavant, les données numériques audio de haute qualité prennent des places énormes pour qu'elles soient stockées.

Or, le taux de compression provient :

- D'une compression avec perte d'informations, réalisée par l'élimination des composants spectraux qui ne jouent pas un rôle fondamental dans la fidélité de la restitution.
- D'une compression sans perte d'informations à travers le codage des données résultants de l'opération de quantification suivant l'algorithme de Huffman [Rol 99].

Nous rappelons que le principe du codage de Huffman consiste à consacrer le mot binaire le plus long aux valeurs les moins fréquentes ; ainsi, une réduction du débit binaire est réalisée. Le processus du décodage extrait les échantillons du flux de bits de MP3. Ce flux consiste en de petites trames qui produisent un son de quelques millisecondes chacune. La trame contient des informations sur le processus de décodage aussi bien que les données audio compressées.

2.1 Format d'entrée du décodeur (trame)

Une trame comprend cinq parties: L'entête (Header), un code détecteur d'erreur CRC (Correcteur Cyclique Redondant), Side information (information propre à la trame), les données principales (main data), et les données auxiliaires (ancillary data). En MP3, chaque trame contient des informations pour produire 1152 échantillons, dont le format est représenté dans la Figure 5.1.

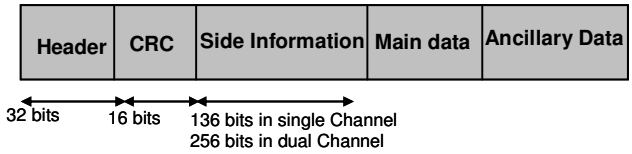


Figure 5.1: Le format d'une trame

-L'entête

la Figure 5.2 représente l'entête qui contient des informations concernant le format de la trame. Elle est composée de 32 bits et commence par un mot de synchronisation (syncword) qui marque le début d'une trame.

L'entête contient aussi des informations sur la trame, y compris le nombre de couches, le débit binaire, la fréquence d'échantillonnage et les paramètres de l'encodage stéréo. Certaines de ces informations, telles que le débit binaire et la fréquence d'échantillonnage, sont nécessaires au décodage alors que d'autres informations, comme le copyright et original bits ne le sont pas.

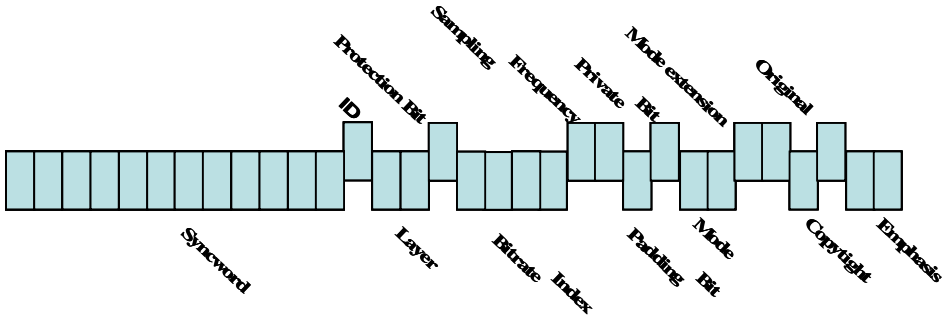


Figure 5.2: Le format d'une entête

-Le CRC

Il est possible de protéger l'entête avec un CRC de 16 bits. Il est optionnel et peut être omis. Quand il est présent, le décodeur doit calculer le CRC dans l'entête et le comparer à celui de la trame. S'ils sont différents, le décodeur doit recommencer la recherche d'un nouveau syncword.

-Side information

Cette partie contient les informations nécessaires au décodage et au traitement de données principales. Ces informations sont globales et utilisées lors des différentes phases du processus de décodage et principalement lors du décodage de Huffman (choix de la table Huffman à utiliser) et de la quantification inverse (les valeurs de correction à appliquer aux bandes de fréquence).

-Données principales

Cette partie contient les facteurs d'échelle et les données codées selon l'algorithme de Huffman. La taille des données codées est variable mais la taille de la trame est toujours la même. Par conséquent, il existe un espace – compris entre la fin des données principales et la fin de la trame – potentiellement inutilisé dans la trame courante. Cet espace est nommé réservoir bit. Les données principales d'une trame illustrées dans la Figure 5.3 peuvent s'étendre sur plusieurs trames antécédentes.

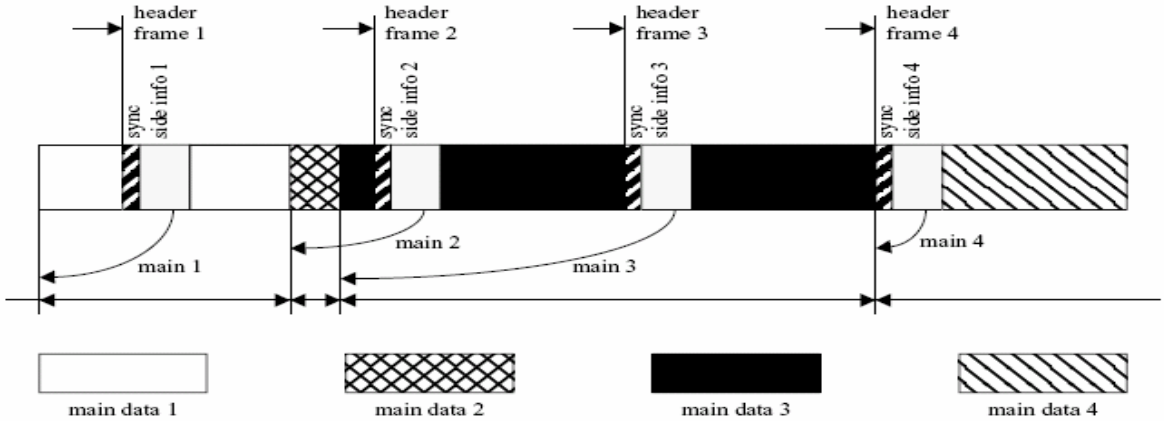


Figure 5.3: Les données principales dans les trames

- **Données auxiliaires**

Elles peuvent contenir des informations définies par l'utilisateur comme par exemple le titre de la chanson, le nom de l'interprète…

2.2 La description de l'algorithme du décodeur MP3

Le processus de décodage permet de retrouver le format audio à partir d'une trame MP3 (bitstream MP3). Les données audio sous forme de bitstream sont d'abord décodées selon l'algorithme de Huffman. Nous obtenons alors les symboles représentant les 576 lignes de fréquence quantifiées et corrigées. La prochaine phase du processus est de faire une quantification inverse de ces symboles afin de retrouver les lignes de fréquences non quantifiées en utilisant les facteurs d'échelle fournis par la trame.

Si la trame contient plus d'un canal d'information audio, le décodeur stéréo recrée les fréquences pour les canaux droit et gauche.

Les dernières phases du décodage produisent des échantillons temporels à partir des lignes de fréquence en appliquant à ces lignes une MDCT inverse (Inverse Modified Discret Cosine Transforme) et un banc de filtre polyphase.

La description du décodeur MP3 est décrite dans la Figure 5.4.

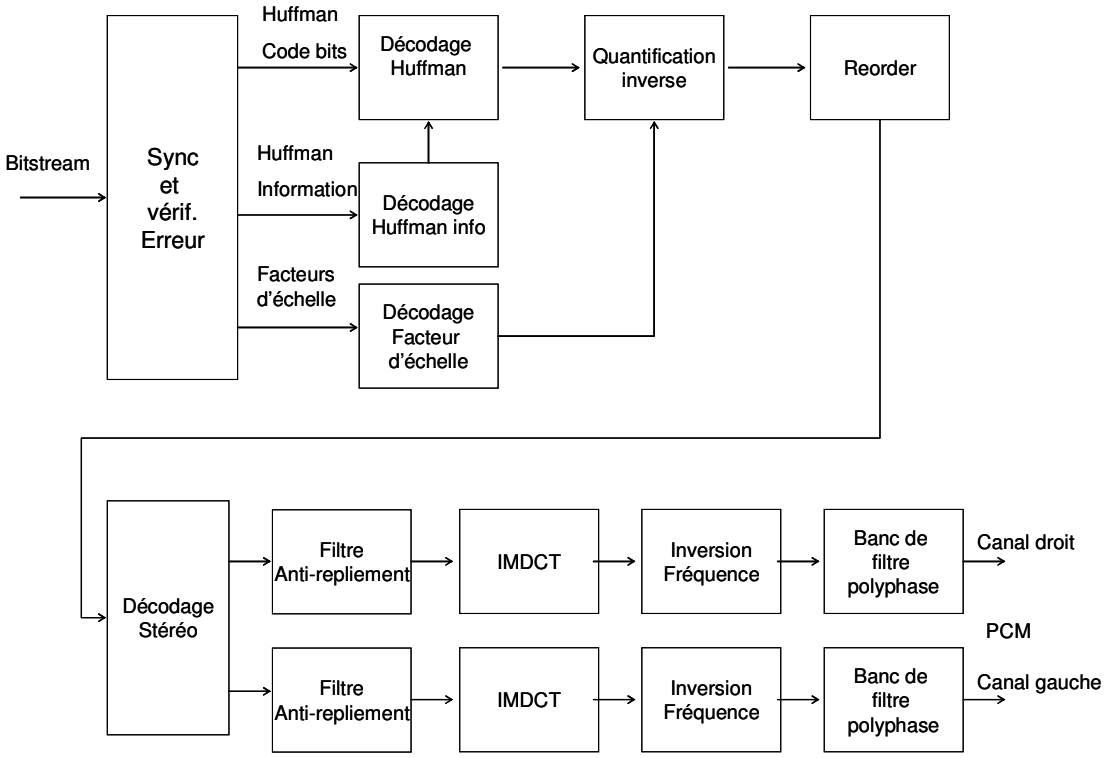


Figure 5.4: Le schéma du décodeur MP3

- **Synchronisation de la trame**

Avant de commencer le processus de décodage, il faut trouver le début de la trame. Un mot de synchronisation formé par une séquence binaire de douze '1' consécutifs, est placé au début de chaque trame.

Il est possible de trouver la même suite binaire à n'importe quelle autre partie de la trame. Dans ce cas le décodeur ne peut savoir s'il s'agit bien du mot de synchronisation ou d'autres données arbitraires. Si la trame contient un CRC, alors il peut être utilisé pour confirmer que nous avons bien trouvé une nouvelle entête valide. Sinon le décodeur calcule la taille de la trame en utilisant les informations de l'entête et détermine ainsi le début de la prochaine trame.

- **Décodeur de Huffman**

La fonction du décodeur Huffman [Rol 99] est de générer à partir des données compressées, les facteurs d'échelle et les 576 entiers représentants les lignes de fréquences originales.

Ces données compressés sont codées selon l'algorithme de Huffman et se trouvent dans la main data.

En MP3, il y a 32 tables Huffman différentes qui sont prédéfinies en se basant sur la récurrence statistique. La side information contient les informations précisant quelle table il faut utiliser.

Le décodeur compare les séquences d'entrée avec les informations présentes dans les tables Huffman. S'il détecte une concordance, le symbole de sortie correspondant est rapporté.

Le décodeur Huffman génère ainsi 576 lignes de fréquences quantifiées représentées par des valeurs entières ainsi que des facteurs d'échelle. Ces facteurs d'échelle sont nécessaires au bloc suivant (quantification inverse) pour rendre à ces lignes de fréquences leurs valeurs originales non quantifiées.

- **Quantification inverse**

Les symboles obtenus lors du décodage Huffman permettent de reconstruire les fréquences originales en utilisant les facteurs d'échelle eux mêmes générés par le décodage Huffman.

Dans le processus du codage, les fréquences sont divisées en un facteur d'échelle et un nombre entier qui est normalement compris entre -15 et + 15. Les facteurs d'échelle sont sauvegardés séparément et les nombres entiers sont codés selon l'algorithme de Huffman. Ils forment ensemble les données principales.

Les lignes de fréquence pour les blocs longs et les blocs courts sont respectivement divisées en 23 et 13 bandes de facteurs d'échelle. Chacune de ces bandes a son propre facteur d'échelle. Le nombre de fréquences dans chaque groupe dépend de la bande de fréquence qu'il représente. Une bande de facteurs d'échelle de basses fréquences contient moins de valeurs qu'une bande représentant les hautes fréquences.

Le processus de quantification inverse utilise les informations globales de la correction trouvées parmi les données de la trame. Ces informations doivent être appliquées à tous les groupes de fréquences. De plus les 21 facteurs d'échelle doivent être appliqués à chacun des groupes indépendamment. Les équations complètes de la quantification inverse pour les blocs courts et les blocs longs respectivement les suivantes :

$$x_i = sg(in_i) \times |in_i|^{\frac{4}{3}} \times 2^{\frac{1(global\_gain[gr]-210-8\times subblock\_gain[window][gr])}{4}} \times 2^{-(scalefac\_multiplier \times scalefac\_s[gr][ch][sfb][window])}$$

$$x_i = sg(in_i) \times |in_i|^{\frac{4}{3}} \times 2^{\frac{1(global\_gain[gr]-210)}{4}} \times 2^{-(scalefac\_multiplier \times scalefac\_l[gr][ch][sfb]+preflag[gr]\times pretab[sfb]}$$

avec :

in : la valeur des symboles générés par le décodeur de Huffman à l'entrée de ce bloc.

x : la séquence de sortie de ce bloc.

sfb : le facteur d'échelle de la sous bande.

window : la fenêtre utilisée dans le bloc court (il y a au total 3 fenêtres).

210 : constante du système définie par ISO/IEC 11172-3 standard, cette valeur assure la quantification appropriée de la sortie.

global gain : la valeur du pas de quantification utilisé pour un canal.

scalefac_multiplier : les facteurs d'échelle ont été quantifiés logarithmiquement lors de l'encodage avec un pas de 2 ou $\sqrt{2}$. La valeur de ce pas est fixée par la valeur du drapeau scalefac_scale. Si ce dernier vaut 0 alors scalefac_multiplier est égal à 0.5 sinon il est égal à 1.

preflag :cette variable est utilisée seulement dans le cas où de longues fenêtres sont appliquées lors de la MDCT. Elle est ajoutée aux facteurs d'échelle Scalefac_l pour une amplification supplémentaire des hautes fréquences.

subbloc_gain : cette variable est utilisée seulement dans le cas où de petites fenêtres sont appliquées lors de la IMDCT. Elle spécifie le facteur de gain à appliquer à chacune de ces petites fenêtres.

Scalefac_s et Scalefac_l : correspondent aux facteurs d'échelle décodés par le décodeur de Huffman et sont utilisés pour ajuster le gain des bandes de facteurs d'échelle.
global_gain, subbloc_gain, scalefac_multiplier et preflag sont donnés par le side information de la trame.

- **Réarrangement (Reorder)**

La fonction de ce bloc est de classer les lignes de fréquence : d'abord par sous bande et ensuite par fréquence.
Durant le processus d'encodage, la MDCT (**Transformée cosinus modifiée discrète)** peut ranger la sortie de deux manières différentes. Normalement, la sortie de la MDCT est rangée en sous bandes par ordre croissant de la fréquence. Cependant, quand un bloc court est décodé une petite fenêtre va être utilisée. Par conséquent la sortie de la MDCT sera classée d'abord par sous bande, ensuite par fenêtres et enfin par ordre croissant de la fréquence. Les différents fenêtrages utilisés par la IMDCT sont illustrés dans la Figure 5.5. Comme les blocs courts sont les seuls à comporter plus d'une fenêtre (ils comprennent trois fenêtres), le reclassement est seulement appliqué aux blocs courts en rangeant les fréquences d'abord par sous bandes et ensuite par fréquences croissantes.
La Figure 5.6 montre le reclassement d'un bloc court en sous bande avec des fréquences croissantes. La nuance de couleur dans chaque sous bande représente les fréquences. La couleur la plus sombre correspond à la fréquence la plus élevée.

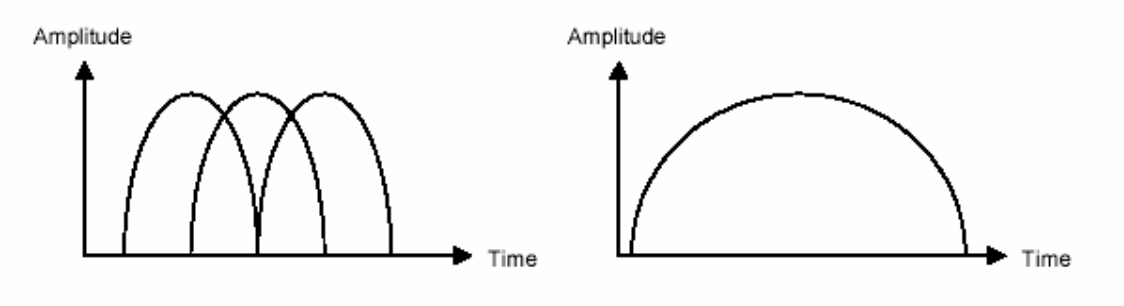


Figure 5.5: Les fenêtres courts et longs utilisés par la MDCT et la IMDCT

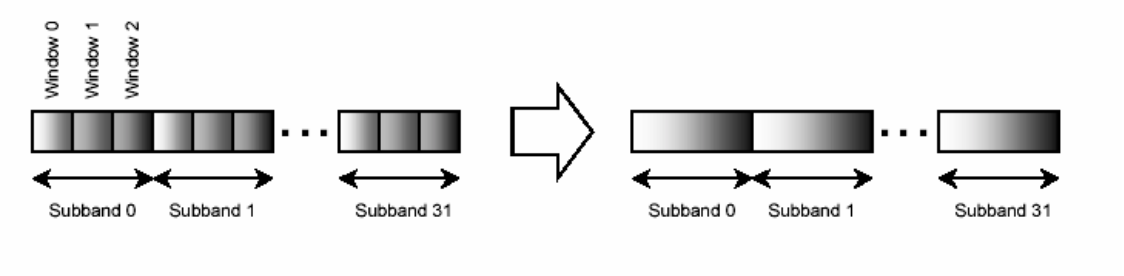


Figure 5.6: L'arrangement des fréquences dans les sous bandes

- **Décodage Stéréo**

En MP3, il existe quatre types différents de canaux : simple canal (mono), double canal (stéréo), middle/side stéréo (MS stéréo) et Intensity stéréo.

Dans ce travail nous avons traité uniquement les deux cas mono et stéréo où les canaux sont traités comme des entités indépendantes.

mono : dans ce cas l'information passe par un seul canal, le canal de droite. Celui de gauche reste inutilisé.

stéréo : dans ce cas chaque canal est traité indépendamment.

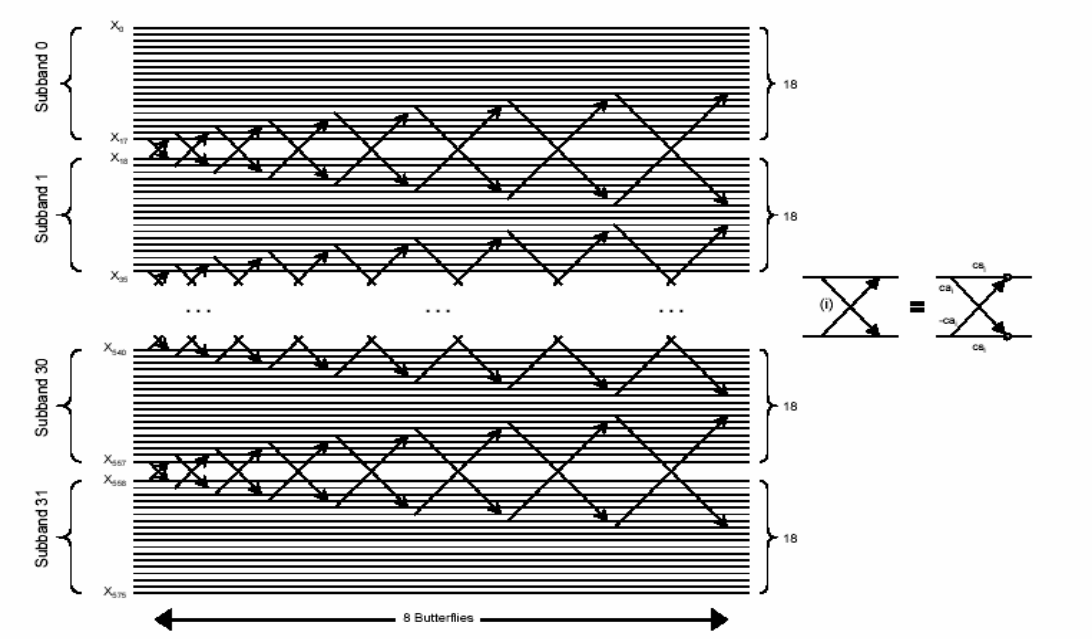


Figure 5.7: Principe du filtre anti-repliement

- **Filtre anti-repliement**

Le bloc anti-repliement a pour fonction de réduire les inévitables effets de recouvrement de spectre introduits par l'utilisation, lors de l'encodage, de filtres passe-bande non idéaux.

Ce filtre anti-repliement fusionne les fréquences en utilisant des calculs de Butterfly pour chaque sous bande. Plus précisément, il applique un filtre de Butterfly sur les 8 dernières lignes de fréquence d'une sous bande et les 8 premières lignes de fréquence de la sous bande suivante.

Les coefficients du filtre Butterfly sont définis dans le ISO/IEC 11172-3 standard. La Figure 5.7 représente le principe du filtre anti-repliement.

- **IMDCT**

La transformée inverse modifiée en cosinus discret (IMDCT) utilisée en MP3 est une 18-points DCT qui produit 36 valeurs à partir de 18 valeurs entrées. Elle ne produit pas plus d'information qu'on lui met en entrée, mais elle disperse le résultat du calcul sur un plus grand nombre de valeurs. C'est pourquoi le terme « modifiée » est employé. La DCT est « inverse » parce qu'elle produit des échantillons de temps à partir des lignes de fréquence.

La transformation du domaine fréquentiel au domaine temporel est accomplie par le bloc banc de filtres polyphase. Au lieu de produire directement des échantillons temporels, l'IMDCT produit à partir des lignes de fréquence d'entrée, des sous bandes d'échantillons pour le banc de filtres. Celles-ci seront employées plus tard pour créer des échantillons temporels.

Les 36 valeurs calculées par l'IMDCT doivent être multipliées par une fenêtre de 36 points avant qu'elles puissent être employées par la prochaine étape dans le processus de décodage. Il existe quatre types différents de fenêtres qui peuvent être appliquées aux échantillons produits. La fenêtre à employer dépend du type du bloc qui se trouve dans la side information.

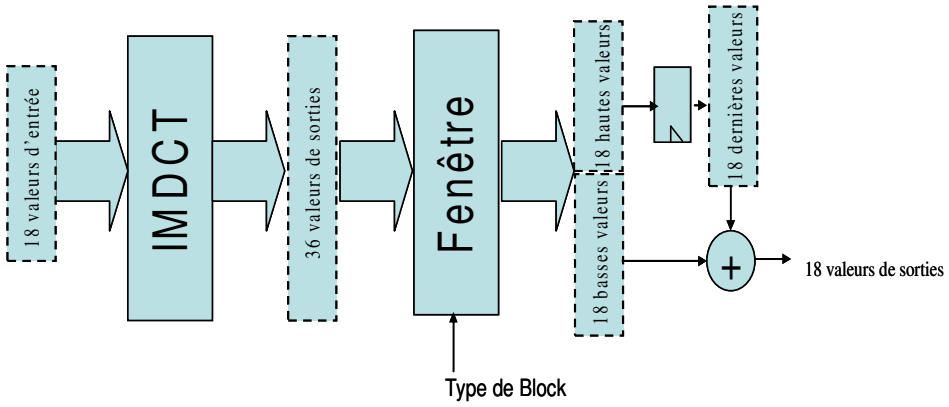


Figure 5.8: Fonctionnement de l'IMDCT et du fenêtrage

La raison de produire 36 valeurs est que les 18 valeurs plus basses sont ajoutées aux 18 valeurs plus élevées de la trame précédente. Le résultat de cette opération est utilisé comme sortie.

Les 18 valeurs les plus élevées sont alors stockées et employées de la même manière dans le décodage de la prochaine trame.

La Figure 5.8 décrit le fonctionnement de l'IMDCT et du fenêtrage.

Le calcul de l'IMDCT est basé sur l'équation suivante :

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos(\frac{\pi}{2n}(2i+1+n/2)(2k+1)), \text{ de } i = 0 \text{ à } n-1$$

Il existe quatre fenêtres différentes qui peuvent être appliquées à la sortie de l'IMDCT. Chaque valeur d'entrée est multipliée par la valeur correspondante de la fenêtre.

- **Inversion de fréquence**

Avant le passage par le banc de filtres polyphase, tous les échantillons impairs dans toutes les sous bandes impaires doivent être multipliés par -1. Ceci est fait afin de compenser l'inversion de fréquence due au banc de filtres.

- **Banc de filtres**

Le banc de filtres polyphase de synthèse, ou la synthèse de sous bande, est la phase finale dans le processus de décodage. Un organigramme complet de la synthèse de sous bandes peut être trouvé dans la Figure 5.9

Il exploite le repliement de spectre et le fenêtrage pour remettre les sous bandes dans le domaine fréquentiel. Ce processus est divisé en deux parties :

1- MDCT :

Les sous échantillons du bloc de transposition sont ordonnés de telle manière que les 32 premières valeurs constituent le premier sous échantillon de chaque sous bande et les 32 prochaines, le deuxième sous échantillon et ainsi de suite. La MDCT traite 32 valeurs à la fois en utilisant les équations suivantes :

$$Y_i = \sum_{k=0}^{31} N_{ik} \times S_k$$

$$N_{ik} = \cos\left(\frac{\pi}{2\times 32}(16+i)(2\times k+1)\right)$$

Les valeurs de sortie Y_i sont stockées dans un tableau.

2- Fenêtrage :

Les valeurs du registre sont multipliées par une fonction porte. Trente deux échantillons PCM sont calculés dans chaque itération. La MDCT et le fenêtrage sont exécutés 18 fois pour chaque granule, ils génèrent à la fin 576 échantillons PCM c'est-à-dire 27 ms à 44.1 KHz.

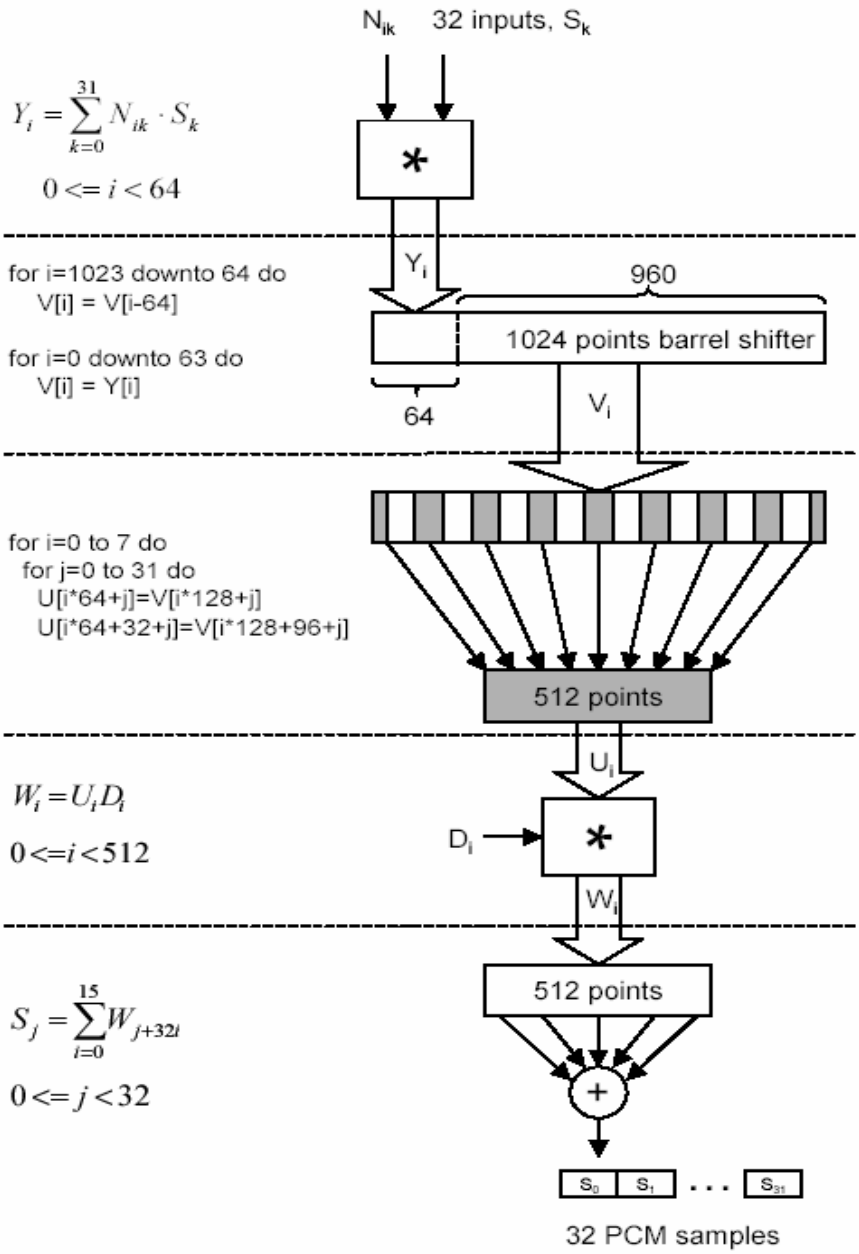


Figure 5.9: Organigramme du filtre de synthèse des sous bandes

3. Le modèle fonctionnel du décodeur MP3 en Simulink

La description fonctionnelle du décodeur MP3 que nous avons modélisé dans l'environnement Simulink, présente l'entrée du flot de conception des MPSoCs proposé au chapitre 3.

Le modèle du décodeur illustré dans la Figure 5.10 est basé sur l'implémentation de plusieurs S-fonctions. Il traite les fichiers MP3 mono et dual stéréo. Nous rappelons que le flux de données, après le décodage de huffman est décomposé en quatre parties (deux canaux et deux granules). Cependant chaque partie subit le même traitement, ce qui permet d'utiliser les mêmes blocs pour traiter les différentes parties du flux.

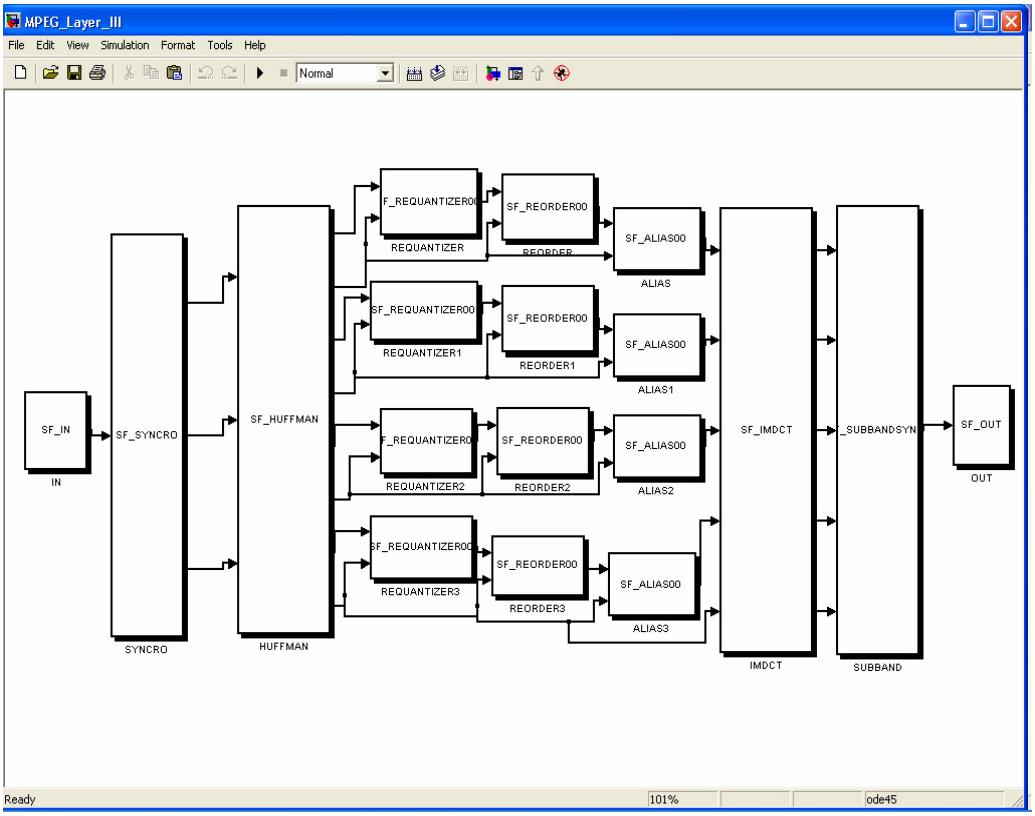


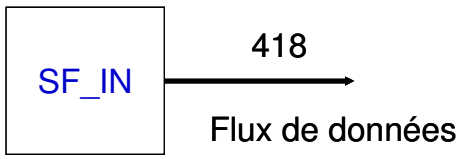
Figure 5.10: Modélisation fonctionnelle du décodeur MP3 dans l'environnement Simulink

Notre spécification fonctionnelle est basée sur l'assemblage et la réutilisation des blocs Simulink. Ces derniers sont conçus pour modéliser les unités fonctionnelles homologues à la représentation algorithmique de l'application. Donc, l'ordre de granularité adopté permet de représenter les fonctions typiques du domaine de traitement de signal. (HUFFMAN, IMDCT, ALIAS, ….etc.). De plus, cette modélisation fonctionnelle explicite bien le parallélisme et le pipeline, ce qui permet d'exploiter des architectures parallèles.

Les différentes unités fonctionnelles utilisées pour spécifier le décodeur MP3 en Simulink sont les suivantes :

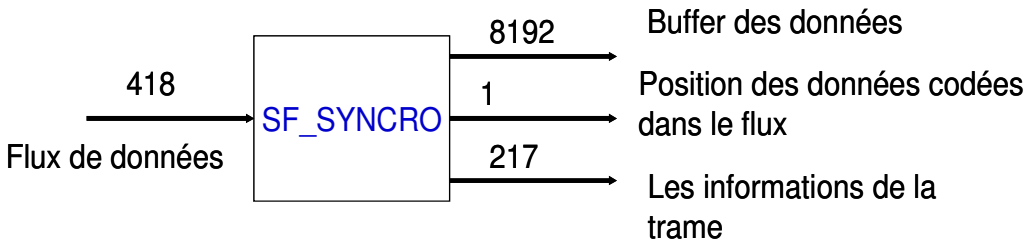
- **SF_IN**

Ce premier bloc lit les données du fichier dans un buffer et les envoie au premier bloc du décodeur. Il représente l'antenne ou le disque de stockage. Chaque fois, il émet un vecteur d'entier de taille 418.



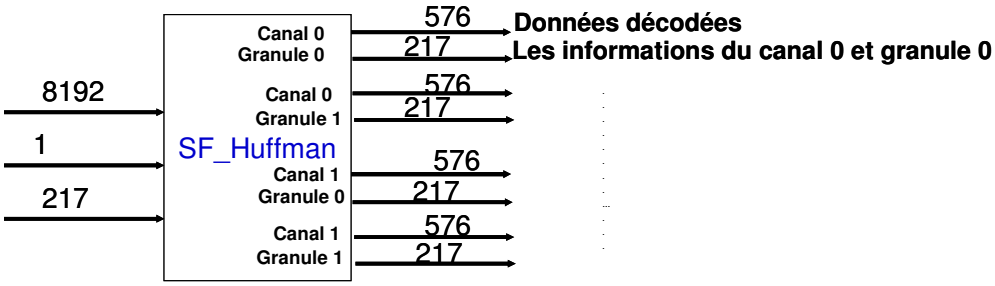
- **SF_SYNCRO**

La détection des trames et l'extraction des informations sont réalisées dans ce bloc. Son entrée est formée d'un vecteur de 418 entiers. Cependant il produit trois sorties. La première est un vecteur d'entier de taille 8192, qui contient les données principales de plusieurs trames concaténées. La taille totale de chaque trame est fixée après l'étape du décodage. La deuxième sortie de ce bloc est un index qui détermine le début des données principales de la trame courante. Ces données doivent être décodées dans le bloc SF_Huffman. La troisième sortie est un vecteur de taille 217 qui représente les informations extraites de l'entête et du 'side information' et qui sont indispensables pour le traitement des données principales de la trame courante dans les étapes suivantes. Ces informations sont stockées sous forme d'une structure de variables pour les transmettre entre les blocs de Simulink, nous avons développé deux fonctions : Une fonction dans le bloc émetteur qui transforme la fonction en un vecteur de 217 entiers et une autre qui se trouve dans le bloc récepteur, pour reconstruire la même structure.



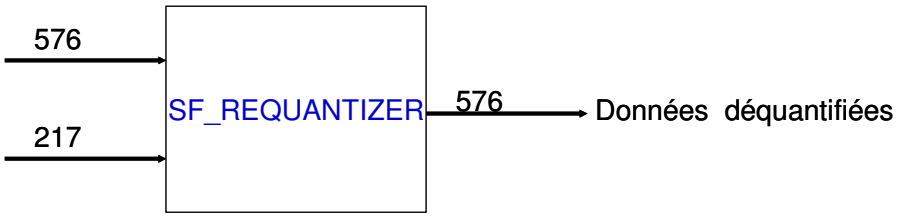
- **SF_HUFFMAN**

Ce bloc décode à l'entrée les données principales audio et les facteurs d'échelle. A la sortie, nous obtenons quatre couples de vecteurs correspondant à une granule et à un canal chacun. Chaque couple est formé par un vecteur de données de 576 entiers et un vecteur d'information composé de 217 entiers correspondant à un canal et une granule.



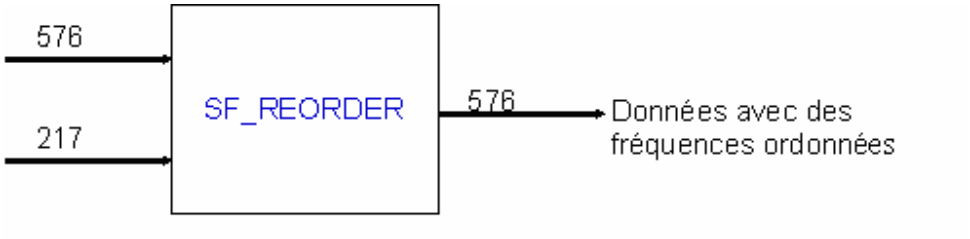
- **SF_REQUANTIZER**

Nous utilisons ce bloc 4 fois dans le modèle. Il prend en entrée les données décodées par le SF_HUFFMAN et le vecteur d'information correspondant à une granule et un canal. Il produit à la sortie un vecteur de 576 entiers déquantifiés.



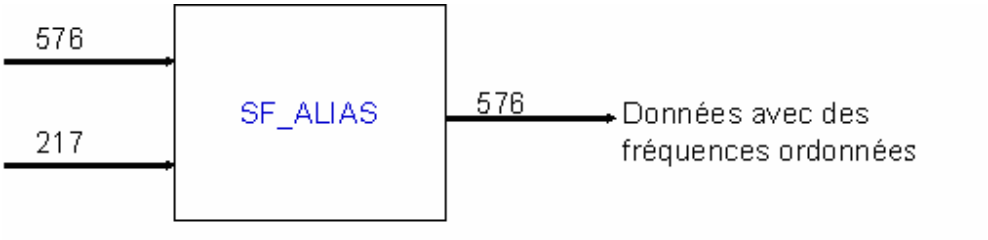
- **SF_REORDER**

Ce bloc est utilisé quatre fois dans le décodeur MP3 implémenté dans Simulink. Il traite les données dans le domaine fréquentiel et produit à la sortie un vecteur de 576 fréquences ordonnées.



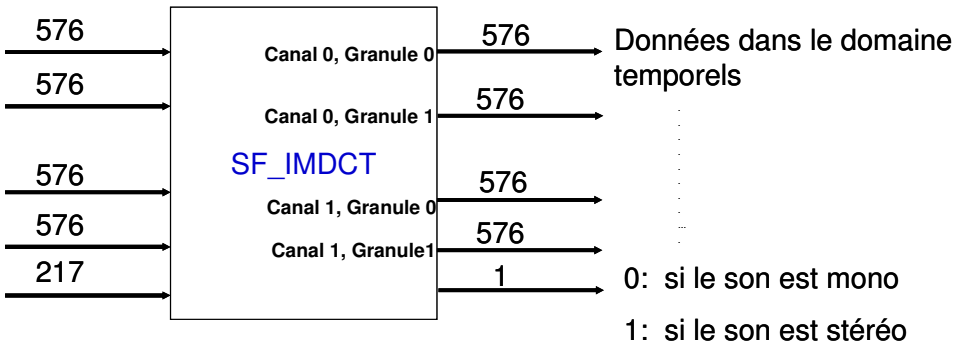
- **SF_ALIAS**

Nous utilisons ce bloc 4 fois. Il produit un vecteur de fréquences formé de 576 entiers. Son rôle consiste à réduire le chevauchement entre ces fréquences.



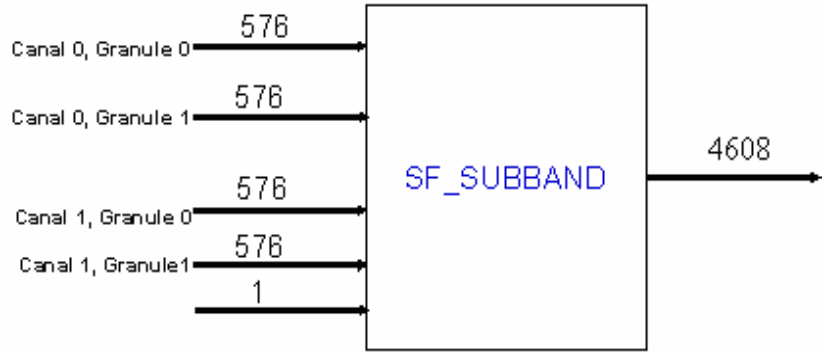
- **SF_IMDCT**

Ce bloc prend en entrée les données produites par les 4 blocs SF_ALIAS et le vecteur d'information propre à la trame. Il produit à la sortie 4 vecteurs de données entiers dans le domaine temporel et un entier indiquant si le son est mono ou stéréo.



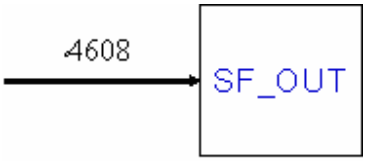
- **SF_SUBBAND**

Ce bloc prend en entrée les quatre vecteurs formés par 576 points temporels et l'entier 1 ou 0 qui indique la nature du son. Il filtre les données et produit un vecteur de 4608 entiers de son décodé ainsi formé de 1152 échantillons.



- **SF_OUT**

Ce dernier bloc reçoit les données de la sortie du décodeur MP3. Il représente l'antenne ou le disque de stockage. Il les écrit dans un fichier audio de sortie.



D'autre part, le 'profiler' de simulation analyse le modèle décrit dans Simulink en générant un rapport. Ce rapport indique le temps de simulation que prend chaque fonction dans un modèle Simulink. Nous remarquons que le bloc SF_IMDCT prends le plus de temps par rapport aux autres blocs ; 16% du temps global. Le bloc SF_Subband prend (12.2%) et le bloc SF_Huffman prend (10.5%).

4. Le modèle transactionnel du décodeur MP3 en Simulink

Un grand écart existe entre la description fonctionnelle du décodeur MP3 et sa description architecturale. Le modèle transactionnel du décodeur est introduit à une phase intermédiaire dans l'environnement Simulink pour établir la continuité entre les modèles du flot de conception et pour accélérer le processus de l'implémentation sur une architecture cible.

En effet, la modélisation transactionnelle dans l'environnement Simulink proposée dans le chapitre précédant permet de combiner dans ce cas l'algorithme et l'architecture du décodeur MP3. Nous décidons d'implémenter l'application sur une plateforme formée par un processeur et deux IPs matériels. Ensuite, la description transactionnelle sera formée par un processeur représenté dans Simulink par un sous système dont le préfixe est 'SW_', et par deux IPs matériels : 'IP_IN', et 'IP_OUT'. Ce processeur englobe seize tâches logicielles représentées dans ce cas par des blocs S-fonctions implémentés en langage C.

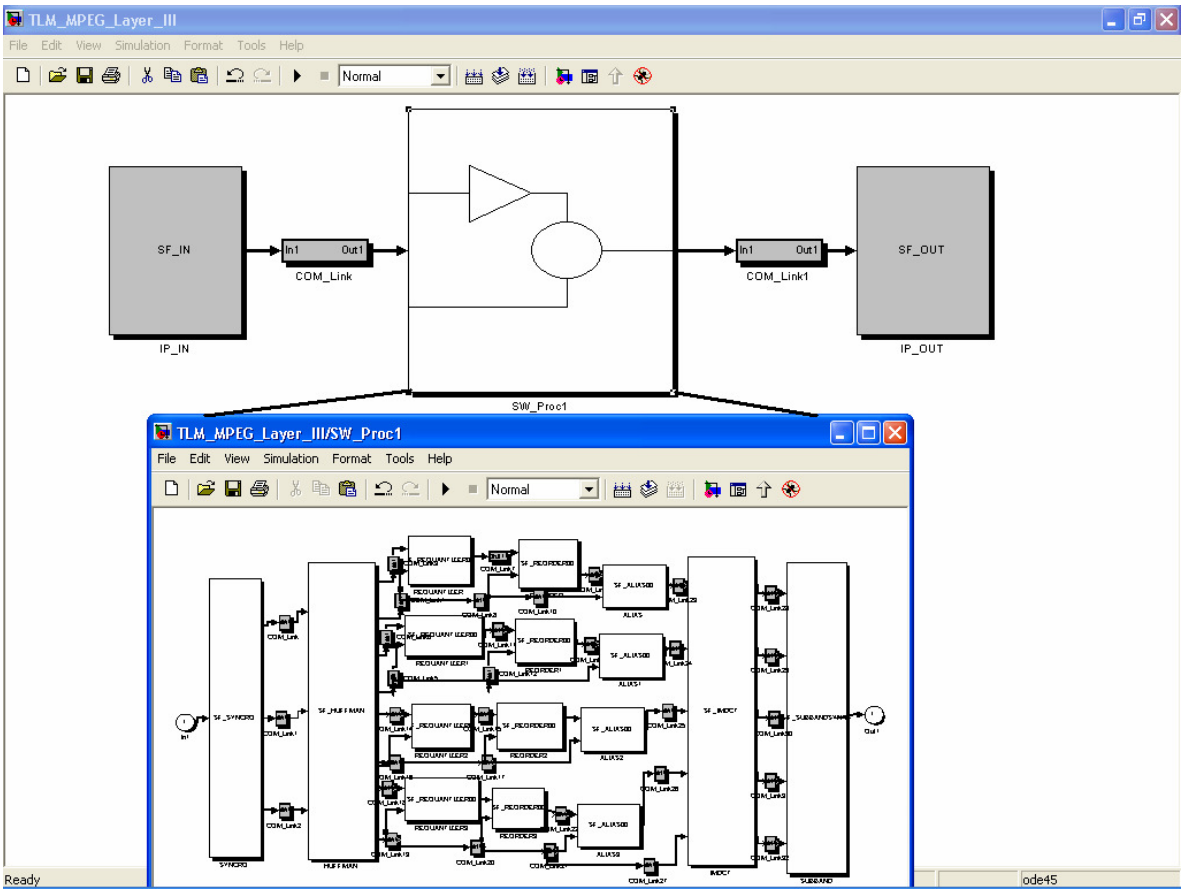


Figure 5.11: Modèle du décodeur de MP3 dans Simulink sur un seul processeur

La communication est établie au moyen des sous systèmes, les noms de ces derniers sont précédés le préfixe ‘Com_’. Les deux canaux de communication entre le processeur et les deux IPs définissent une topologie ‘point à point’ entre les trois composants. Néanmoins, la communication intra processeur est formée par trente quatre canaux de communications dont les rôle, est de transférer les données entre les différentes tâches. Ces différents canaux de communication peuvent être implémentés par différents types de communication. Ainsi nous pouvons valider les protocoles de communications et l’interaction entre les différents composants. La Figure 5.11 illustre le modèle transactionnel du décodeur MP3 dans l’environnement Simulink.

5. Génération du décodeur MP3 en Colif

Le générateur de la macro architecture développé au sein de cette thèse permet de lier l'outil d'exploration d'algorithme Simulink aux outils de génération d'architecture ROSES et Macro_cell builder. Il prend en entrée le décodeur MP3 implémenté au niveau transactionnel en Simulink. Ainsi, plusieurs étapes se succèdent pour générer le décodeur MP3 en Colif.

En premier temps, c'est la génération de l'arbre intermédiaire où seront stockées toutes les données correspondantes au modèle Simulink, (composants, interconnections, …etc.). Cet arbre formé par plusieurs objets (système, bloc, ligne, branche) permet après son analyse de générer le décodeur en Colif. Ce dernier constitue la plateforme, le squelette de l'architecture du décodeur. Toutefois, il manque la sémantique (les paramètres de l'architecture) et le comportement qui seront générés dans les étapes de conception suivantes.

La création de Colif consiste à générer des modules, des interfaces et des interconnections. Dans le cas du décodeur MP3, la structure Colif générée est formée par :

- 16 modules simples, et 3 modules virtuels.
- 37 ports d'entrée simples, 28 ports de sortie simples, 2 ports d'entrée virtuels et 2 ports de sortie virtuels.
- 33 interconnections simples et 2 interconnections virtuelles.

La Figure 5.12 ci-dessous illustre la structure du décodeur MP3 en Colif.

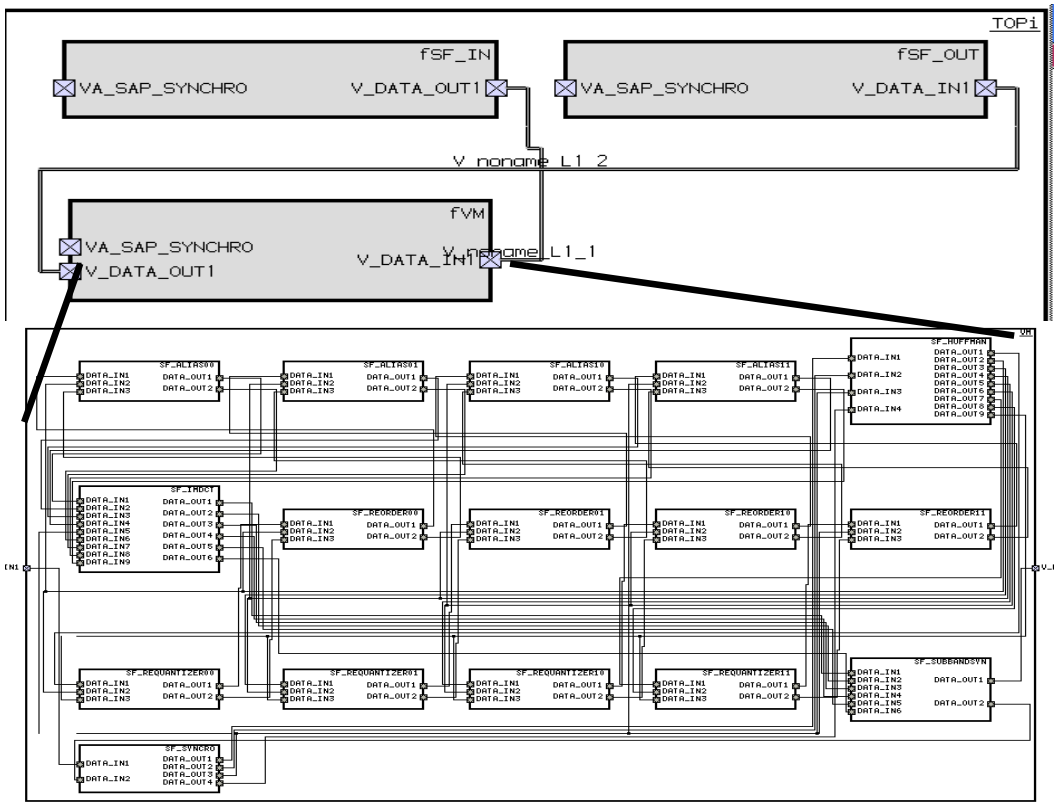


Figure 5.12:Le décodeur MP3 généré dans COLIF à partir de Simulink

6. Spécification des paramètres et raffinement de l'architecture du décodeur MP3

Cette phase permet de produire à partir de l'arbre intermédiaire du décodeur MP3 des fichiers de paramètres génériques que requiert l'architecture. Ces paramètres sont indispensables pour simuler l'architecture virtuelle et pour synthétiser et raffiner le modèle architectural. Ils présentent la sémantique du langage intermédiaire Colif. Ils seront utiles pour déterminer les interfaces de co-simulation, les interfaces logicielles et les interfaces matérielles à partir des librairies prédéfinies.

Ainsi pour l'application du décodeur MP3, deux fichiers de paramètres génériques sont générés. Le premier correspond aux trois modules virtuels et aux interconnections représentant un processeur, deux IPs matériels, 2 canaux de transfert de données et des interfaces de communication.

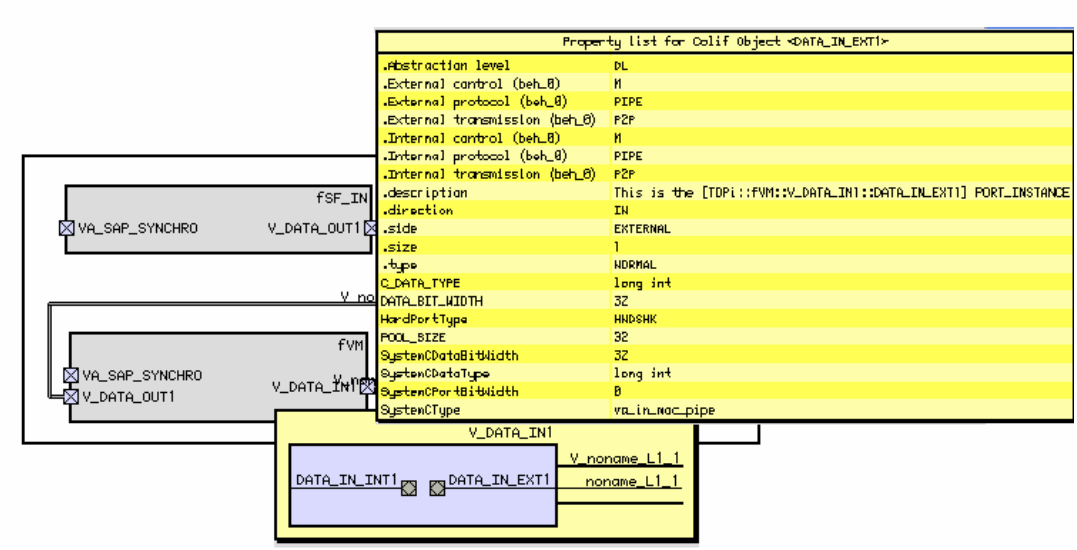


Figure 5.13: Les paramètres du port d'entrée externe du processeur qui décode le son MP3

Le deuxième fichier décrit les paramètres propres aux différentes tâches encapsulées par le processeur, aux interfaces et aux interconnections intra processeur. Ces fichiers générés sont ensuite personnalisés par le concepteur qui connaît bien l'architecture. Les valeurs de ces paramètres sont définies (type de processeur, adresses mémoires, protocoles, interruptions, ….etc.) en remplaçant la valeur 'xxx' dans les fichiers par la valeur qui convient à l'architecture.

Après cette étape, les paramètres sont extraits à partir de ces 2 fichiers personnalisés et à partir des fichiers propres aux protocoles de communication situés dans la bibliothèque. L'ensemble est importé dans la structure Colif du décodeur MP3 pour ajouter la sémantique aux différents éléments de l'architecture. Le raffinement se poursuit par l'ajout d'une tâche 'standby' à l'intérieur du processeur.

En conséquence, nous apercevons un modèle du décodeur MP3 au niveau macro architecture annoté. La Figure 5.13 représente les paramètres du port d'entrée externe du processeur. Enfin, la transformation des fichiers S-fonctions.c de Simulink en des tâches SystemC est réalisée en conservant le même code C, et en ajoutant des APIs de communication correspondant aux protocoles déjà définis. Ainsi, nous transformons 9 S-fonctions en 9 tâches SystemC.

Le décodeur MP3 au niveau macro architecture est simulé par cosim_x en générant des adaptateurs de simulation. Nous constatons que le processus de décodage à ce niveau donne les mêmes résultats de la simulation au niveau Simulink.

7. Génération de la Micro architecture

La génération de la micro architecture est produite à partir de la description macro architecture du décodeur MP3. Les détails de l'architecture sont rendus explicites par l'intermédiaire des outils de synthèse. D'une part, ASAG raffine le module virtuel qui représente le processeur.

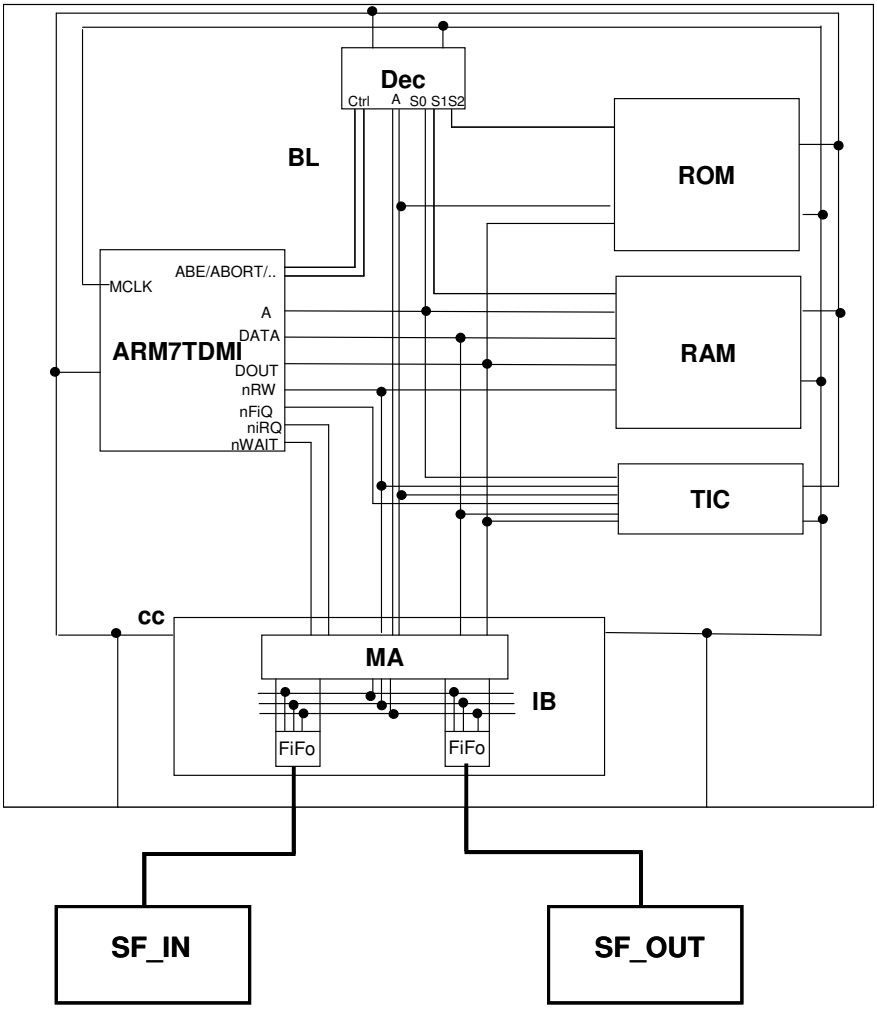


Figure 5.14: Le décodeur MP3 implémenté au niveau micro architecture

Ce raffinement est achevé en ajoutant des composants à partir de sa propre bibliothèque suivant les paramètres de l'architecture définis par le concepteur dans l'étape précédente. D'autre part, le système d'exploitation qui ordonne les différentes tâches et qui gère les entrées et les sorties du processeur, est généré par l'outil ASOG. La Figure 5.14 représente le décodeur MP3 au niveau micro architecture. La simulation de ce système est accomplie au moyen de l'outil Cosim-x.

8. Comparaison entre les différents niveaux d'abstraction du décodeur MP3 à travers le flot de conception

En utilisant plusieurs outils développés et existants nous aboutissons à une implémentation au niveau micro architecture du décodeur MP3. La description au niveau Simulink qui comprend l'exploration d'algorithme et la simulation au niveau système du décodeur MP3 prend 3 fois moins de temps que nécessite la modélisation et la validation au niveau macro architecture. La différence est due à l'environnement de Simulink; à la facilité de déboguage et à l'abstraction de la communication par rapport à la description au niveau macro architecture. La modélisation à ce dernier niveau est plus complexe avec les différents paramètres qu'il faut ajouter.

Le Tableau 5.1 récapitule le temps de simulation et de Co-simulation mesuré aux différents niveaux d'abstraction proposés à travers le flot de conception. Nous simulons le décodeur MP3 en Simulink, au niveau macro architecture, et au niveau micro architecture (simulation native avec un OS à l'aide de UNIX et avec un ISS). Vu la vitesse de simulation à 4 niveaux d'abstraction, nous constatons que la simulation du modèle en Simulink est 50 fois plus rapide que celle au niveau macro architecture. La différence est principalement due à la complexité de la description et aux détails de la communication qui sont présents au niveau macro architecture. La validation avec une simulation native en présence d'un OS à l'aide d'un serveur UNIX est 100 fois plus rapide que la simulation au niveau ISA (Instruction Set Architecture). Cette différence démontre clairement les avantages de la phase de validation aux différents niveaux d'abstraction. A chaque niveau une partie du SoC est validée progressivement. Au niveau Simulink, l'algorithme de l'application est validé. Au niveau macro architecture et dans

l'environnement SystemC, la communication et l'interaction entre les tâches sont validées.

Le niveau de description du décodeur MP3	**Le temps de simulation pour décoder 4s. de son MP3**
Simulink	5s.
Macro architecture	56s.
Native OS-Unix	35 mn.
ISA (Instruction set Architecture)	55h.

Tableau 5.1: Temps de simulation et co-simulation du décodeur MP3

Au niveau micro architecture, la simulation native valide l'application écrite au-delà d'une couche de système d'exploitation OS et finalement l'architecture globale est validée par l'intermédiaire d'une Co-simulation au cycle précis avec un ISS. Nous constatons que le temps de décodage d'une seconde du son MP3 requière 100 Mégacycles. Afin de décoder en temps réel nous avons besoin d'un système utilisant une horloge de 100 MHZ. Notons que la taille totale du code binaire inclus le OS est de 66.55 KO et la taille des données est de 11 KO.

9. Conclusion

Nous avons proposé un flot de conception de haut niveau des MPSoCs à partir de Matlab\Simulink. C'est un travail novateur parce que les travaux récents à partir de cet environnement visent uniquement la conception des ASICS ou bien le ciblage du logiciel sur un processeur unique.

Dans ce chapitre nous avons présenté une application multimédia formée par plusieurs fonctions typiques de traitement du signal ; le décodeur MP3. Il est utilisé pour

illustrer la méthodologie et le flot de conception des systèmes sur puce proposés au cours de cette thèse. Ce décodeur est spécifié au niveau fonctionnel en plusieurs blocs S-fonctions en langage C dans l'environnement Simulink. Ensuite, cette application est modélisée au niveau Simulink transactionnel où le système est partitionné en des parties matérielles/logicielles, et les communications sont définies entre les différents composants. L'algorithme et l'architecture sont ainsi combinés dans un seul modèle. L'outil GMA (Générateur de la macro architecture) a intervenu pour transposer le modèle transactionnel du décodeur MP3 en une architecture virtuelle dans le langage intermédiaire Colif. Ce même outil a permis d'additionner les différents paramètres qui représentent la sémantique de la description architecturale du décodeur MP3. L'architecture ainsi annotée est synthétisée par les différents outils du flot de conception. Elle est également simulée à plusieurs niveaux d'abstraction.

L'analyse du décodeur MP3 à partir d'un modèle fonctionnel de haut niveau à travers les différentes étapes de raffinements matériels\logiciels a prouvé l'efficacité et l'intérêt de notre approche.

CHAPITRE 6

CONCLUSIONS ET PERSPECTIVES

Sommaire

1. Conclusions

Les SoC deviennent de plus en plus complexe. Cette complexité croissante est accentuée par l'émergence de nouvelles applications télécoms et multimédia avec des contraintes fonctionnelles de plus en plus sévères (puissance de calcul, consommation, d'embarquabilité, reconfigurabilité) et non fonctionnelles (temps de mise sur le marché, rétrécissement de la durée de vie du produit, coût, …). Pour répondre à ces exigences et maîtriser cette complexité de nombreux travaux de recherche et de développement sont menés aussi bien sur les aspects architecturaux et technologiques que sur les aspects méthodologies et outils de conception. Concernant les architectures et technologie de fabrication, l'intégration de dizaines, voir des centaines de cœurs de traitement autour d'un réseau de communication sophistiqué sur une seule puce domine les axes de recherche actuels pour répondre aux contraintes de ces nouvelles applications télécoms et multimédia. Il s'agit donc d'une évolution des architectures des SoC vers des architectures multiprocesseurs avec des réseaux de communication complexes, les MPSoC et NoC. Concernant les méthodologies et les outils de conception, le défi est de maîtriser la complexité en remontant le niveau d'abstraction au plus haut niveau et en automatisant le flot du passage de la spécification algorithmique à l'implémentation validée et l'intégration du système complet matériel/logiciel.

Dans le cadre ce thèse nous avons abordé le problème de la conception de haut niveau des systèmes multiprocesseurs monopuces. Plus précisément, nous avons élaboré des approches et des stratégies de conception permettant de résoudre des problèmes liés à la conception de haut niveau : réduire le gap entre le modèle au niveau système et les modèles d'architectures, et résoudre le problème de la discontinuité entre les outils de conception et d'exploration des algorithmes (Simulink/Matlab) et les outils de conception des architectures (Roses et Macrocell Builder).

Nous avons commencé nos travaux, d'une part par l'analyse des caractéristiques et des spécificités des systèmes logiciel/matériel embarqués sur puce, et d'autre part par une étude approfondie des méthodologies et des flots de conception existants spécifiques à notre domaine. Cette étude nous a montré que la conception des systèmes multiprocesseurs monopuces doit se baser sur des approches de conception mixtes

(descendante et ascendante). C'est-à-dire aussi bien sur l'assemblage et la réutilisation de composants existants préconçus et prévalidés et d'autre part sur la modélisation système et l'augmentation du niveau d'abstraction. Cependant, ceci engendre plusieurs problèmes lors de la démarche de conception des MPSoCs. Dans le cadre de cette thèse nous nous sommes focalisés sur trois problèmes qu'ils n'étaient pas résolus (ou partiellement non résolus). Le premier problème est relatif à la spécification système. Pour modéliser un système, il faut d'abord faire des choix pertinents : environnements et langages de programmation, les modèles de calcul, les modèles de communication et de synchronisation, … Le deuxième problème concerne le « gap » et la « discontinuité de modèles » qui peut exister, entre la spécification système et le niveau implémentation. Le troisième problème est lié directement au deuxième problème et concerne l'automatisation des étapes de raffinement. Le passage manuel de la spécification à l'implémentation est ardu et requiert des compétences multidisciplinaires. Il peut-être source d'erreurs complexes difficilement détectable (composition de l'application parallèle, logiciel système et le matériel multiprocesseur). Ce passage n'est plus applicable à partir d'un certain degré de complexité.

Pour la modélisation fonctionnelle en Simulink, nous avons défini un sous-ensemble de Matlab/Simulink et nous avons associé un ensemble de règles de description (concernant le niveau de parallélisme, l'abstraction des données, le pipeline, le contrôle) permettant la spécification et la validation fonctionnelle efficace des algorithmes de l'application.

Pour réduire le « gap » entre le modèle fonctionnel et le modèle d'architecture en SystemC, nous avons proposé un nouveau model intermédiaire transactionnel exécutable dans l'environnement Simulink qui combine à la fois l'algorithme et l'architecture dans un même modèle de représentation. Il permet d'une part l'exploration rapide d'architecture dans l'environnement Simulink et d'autre part de préserver la « continuité » entre le niveau système et le niveau architecture. Il permet ainsi d'établir un lien entre les outils de conception des algorithmes et les outils de conception des architectures.

Nous avons développé un outil permettant la génération automatique de la macroarchitecture à partir du modèle mixte algorithme/architecture transactionnel en Simulink. Cette macro architecture est le point d'entrée des outils de conception

d'architecture : Roses pour les interfaces matériel/logiciel et MacroCell Builder pour la conception du matériel dédié. Cet outil permet ainsi de grouper tous les outils dans un flot complet qui part de la spécification fonctionnelle de l'algorithme jusqu'à l'intégration du système complet matériel/logiciel. Il garanti, ainsi, une continuité dans la méthodologie de conception de haut niveau.

Cette méthodologie a été validée sur une application en vraie grandeur, « codec standard MPEG Layer III ». Nous avons présenté et retracé toutes les étapes importantes qui nous ont conduit à la validation de l'approche sur cette application, avant l'étape finale de l'intégration du système MPSoC matériel/logiciel. Ces étapes sont : la spécification fonctionnelle parallèle et « pipelinée » sous Matlab/Simulink; le découpage des algorithmes en blocs réutilisables et paramétrables intégrant des « *S-functions* »; la description du modèle intermédiaire TLM et l'exploration algorithme/architecture ; la génération de macro architecture, la génération des interfaces matériel/logiciel ; la génération de micro architecture ; et la simulation et la validation à tous les niveaux d'abstractions jusqu'au niveau RTL.

2. Perspectives

Nous pensons que les concepts du modèle TLM au niveau Simulink mérite d'être formalisés et qu'une étude spécifique sur le modèle proposé peut être approfondie. D'un point de vue pratique, une continuité de ce travail consiste à étendre le modèle transactionnel dans Simulink pour estimer les performances et explorer l'espace des solutions architecturales à ce niveau. Ainsi, l'architecture adéquate sera choisie d'une façon rapide pour répondre aux contraintes déjà définies. Aussi l'ajout des techniques d'optimisation du code en vue de réduire la taille mémoire et/ou la consommation peut aussi être une perspective.

Bibliographie

[Bag 02'] A. Baghdadi, "Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques", Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2002

[Buc 94] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, Int. Journal of Computer Simulation, special issue on Simulation Software Development, vol. 4, pp. 155-182, April, 1994.

[Bag 02] A. Baghdadi, et al., "Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems" ," *IEEE Trans. on Software Engineering*, vol. 28, no. 9, pp. 822-832, Sept. 2002.

[Bou 06] A. Bouchima " Modélisation du logiciel embarqué à différents niveaux d'abstraction en vue de la validation et la synthèse des systèmes monopuces ", document de thèse, Juin 2006

[Bra 00] K. Brandenburg and H. Popp, "An Introduction to MPEG Layer-3", Fraunhofer Institute, EBU Technical Review, June 2000.

[Cos 00] P. Coste, F. Hessel, A. Jerraya. Multilanguage codesign using SDL and Matlab, 2000.

[Ces 01] Cesario W., Nicolescu G., Gauthier L., Lyonnard D., "Colif: a Multi Design Representation for Application-Specific Multiprocessor System-on-Chip Design", 12th Rapid System Prototyping (RSP), California, 2001.

[Ces 02] W. Cesário, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, “Component-Based Design Approach for Multicore SoCs”, Design Automation Conference (DAC), New Orleans, 2002

[Cai 03] Cai L., Gajski D. Transaction Level in System Level Design, CECS Technical Report 03 – 10 Mar 28, 2003, University of California, Irvine.

[Cat] Catapult C synthesis for ASIC and FPGA http://www.mentor.com/

[Dav 02] W. Rhett Davis, et al., “A Design Environment for High Throughput, Low Power Dedicated Signal Processing Systems”, IEEE JOURNAL OF SOLID STATE CIRCUITS, VOL. 37, NO. 3, MARCH 2002

[Dia 97] A. Diagne. Systèmes répartis et coopératifs : une approche multi-formalismes de spécification de systèmes répartis : transformations de composants modulaires en réseaux de petri, Thèse de doctorat, Université Paris 6, 1997.

[Dem 97] G. De Micheli, R. Gupta. Hardware/Software Co-design, Proceedings of the IEEE, Vol.85, N.3, pp.349-365, 1997.

[Edw 97] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", Proceedings of the IEEE, Vol. 85, No. 3, March 1997.

[Gha 03] Ferid Gharsalli, “Conception des interface logiciel- matériel pour l’intégration des mémoires globales dans les systèmes monopoces”, Thèse de doctorat, INPG, Spécialité Informatique, laboratoire TIMA, 2003

[Gra 04] Grasset A., Rousseau F., Jerraya A.A. “Network Interface Generation for MPSOC: from Communication Service Requirement in RTL Implementation”, 15th Rapid System Prototyping (RSP), 2004

[Gau 01] Lovic Gauthier, “Génération de système d’exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques“, Thèse de Doctorat INPG, Spécialité Microélectronique, laboratoire TIMA, 2001

[Hav 02] 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.

[Hil 93’] P. N. Hilfnger. Silage reference manual, draft release 2.0. (1993).

[Hil 93] CM, “The CM-5 Connection Machine: A Scalable Supercomputer”, W. Daniel Hillis and Lewis W. Tucker., Communications of the ACM, November 1993, Vol. 36, No. 11.

[Inn] Innoveda, “eArchitect,”available at http://www.innoveda.com/

[Iso 93] ISO/IEC 11172-3, Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio, 1993.

[Jan 01] J. W. Janneck, E. A. Lee, J. Liu, X. Liu, S. Neuendorer, S. Sachs, Y. Xiong. Disciplining heterogeneity : the Ptolemy approach. In ACM SIGPLAN workshop on langages compilers and tools for embedded systems LCTES 2001 Snoowbird Utah, june 2001.

[Jan 02] R. Janka, L. Willis, L. Baumstark, "Virtual Benchmarking and Model Continuity in Prototyping Embedded Multiprocessor Signal Processing Systems," *IEEE Trans. on Software Engineering*, vol. 28, no. 9, pp. 832-846, Sept. 2002.

Jea [06] J. P. DIGUET, et al., " EPICURE : A partitionning and co-design framework for reconfigurable computing » *ELSEVIER.* Microprocessors and Microsystems 30 (2006) Pages : 367-387.

[Jer 04'] A. A. Jerraya and W.Wolf, editors. "Multiprocessor System-on-Chips". Morgan Kaufmann Publishers Inc., Octobre 2004.

[Jer 04] A.A. Jerraya, "Long Term Trends for Embedded System Design", EUROMICRO Symposium on Digital System Design (DSD 2004), Rennes, France, Septembre. 2004.

[Lyo 03] Damien Lyonnard, "Approche d'assemblage systématique d'élément d'interface pour la génération d'architectures multiprocesseurs", Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2003

[Lee 03] E. A. Lee. Overview of the Ptolemy project. Technical Memorandum No. UCB/ERL M03/25,University of California, Berkeley, CA, 94720, USA, July 2003.

[Liu 01] Modeling distributed hybrid systems in Ptolemy II He Liu; Xiaojun Liu; Lee, E.A.; American Control Conference, Proceedings of 2001, Volume: 6, Page(s): 4984 - 4985.

[Lee 06] E. A. Lee and H. Zheng, "HyVisual: A Hybrid System Modeling Framework Based on Ptolemy II", in *IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'06)*, Sardinia, June 7-9, 2006.

[Nic 02] Eugenia G. N. Nicolescu, "Spécification et validation des systèmes hétérogènes embarqués" , Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2002

[Nee 00] S.Neema. System-level synthesis of adaptive computing systems, Mar.2000.

[Mat] Matlab/Simulink, *http://www.mathworks.com/*

[Moo 97] V. J. Mooney III, G. De Micheli. Real time analysis and priority scheduler generation for hardware- software systems with a synthesized run-time system. In Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design, pages 605-612. IEEE Computer Society, 1997.

[Pto] Ptolemy, University of California, Berkeley, USA, http://www.ptolemy.eecs.berkeley.edu/ptolemyclassic/body.html.

[Pav 04] Yanick Paviot, "Implémentations mixtes logicielles/matérielles des services de communication pour l'exploration du partitionnement logiciel/matériel", Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2004

[Ron 99] Ron W., "Is SoC really different?" EETIMS November 8, 1999. http://www.eetimes.com/story/OEG19991108S0009

[Rea] Real-Time Workshop, http://www.mathworks.com

[Rea'] RealChip Custom communication Chips, Systems-on-Chips. http://www.realchip.com/Systems-on-Chips/systems-on-chips.html

[Rol 99] Rolf Johannesson and K. Sh Zigangirov ,*"Fundamentals of Convolutional Coding"*,IEEE; ISBN: 0780334833, March 1999.

[Sor 05] Y. Sorel, et al. "Rapid prototyping for heterogeneous multi- component Systems: An MPEG4 stream over an UMTS link", EURASIP Journal of ASP 2005.

[Sto 95] E. Stoy: A Petri Net Based Unifed Representation for Hardware/Software Co-Design, LicentiateThesis, LiU-Tek-Lic 1995: 21, Dept. of Computer and Information Science, Linkping University,1995.

[Sch 00] S. Schulz and J. Rozenblit. Concepts for model compilation, proceedings of ICDA Conference, 2000.

[Sim] Simulink, http://www.mathworks.com/

[Sem 01] L. Semeria, K. Sato, and G. De Micheli, "Synthesis of Hardware Models in C With Pointers and Complex Data Structures " *IEEE Transactions on VLSI Systems Vol. 9, pp. 743-756,* Dec. 2001.

[Syn 03] Synopsys Inc, "*System Studio",* product datasheet at www.synopsys.com, 2003.

[Var 02] M. Varea. Mixed control/data-fow representation for modelling and verification of embedded systems. Technical report, University of Southampton, Mar. 2002.

[Vsi 00] VSIPL Forum, "VSIPL v1.0 API Standard Specification,"available at http://www.vsipl.org/PubInfo/pubdrftrev.html, Mar. 2000.

[Wak 00] K. Wakabayashi, T. Okamoto, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective", *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12), December 2000.

[Xil] Xilinx/Matlab System generator: http://www.xilinx.com/

[Zer 04] N. Zergainoh, et al., "Matlab based Environment for designing DSP Systems using blocks", *12th Workshop on Synthesis And System Integration of Mixed Information technologies, (SASIMI'04)* Japan.

[Zer 05] N. Zergainoh, et al. "IP-Block-based Design Environment for High-Throughput VLSI Dedicated Digital Signal Processing Systems", ASP-DAC 2005 proceedings, January 2005, Shanghai, China.

Publications

Y. ATAT, N. E. ZERGAINOH « Simulink- based MPSoC Design: New Approach to bridge the Gap between Algorithm and Architecture Design**» IEEE Computer Society Annual Symposium on VLSI, Porto Alegre, Brazil (To appear May 2007)**

Y. ATAT, N. E. ZERGAINOH, A. A. JERRAYA « Conception des Systèmes sur puce à partir de Matlab\Simulink » **8ème Journées Nationales en Microélectronique 2005 Paris – France**

Y. ATAT, N. E. ZERGAINOH, A. A. JERRAYA. « Environnement de Conception, de Validation, et de Prototypage Rapide des Systèmes Multiprocesseurs sur Puce, Pour les applications de traitement de signal » **9ème Journées Nationales en Microélectronique 2006 Rennes –France**

Y. Y.ATAT, N. E. ZERGAINOH, A. A. JERRAYA « Conception de haut niveau des MPSoCs à partir d'une spécification Simulink : Passerelle entre la conception au niveau Système et la génération d'architecture » **TSI Technique et Science informatiques (en cours de préparation)**

RESUME

La technologie de fabrication actuelle permet l'intégration d'un système multiprocesseur complexe sur une seule pièce de silicium (MPSoC pour Multiprocessor System-on-Chip). Une façon de maîtriser la complexité croissante de ces systèmes est d'augmenter le niveau d'abstraction et d'aborder la conception au niveau système. Cependant, l'augmentation du niveau d'abstraction peut engendrer un fossé entre les concepts au niveau système et ceux utilisés pour l'implémentation de l'architecture Matériel/Logiciel du MPSoC.
L'objectif de cette thèse est de combler le gap entre les deux niveaux d'abstractions utilisés en proposant une passerelle efficace entre les outils d'aide au développement d'algorithmes (Matlab\Simulink) et les outils de conception des architectures (ROSES et macro-Cell builder). Ceci est accompli :
- En définissant un modèle intermédiaire transactionnel dans l'environnement Simulink. Ce modèle intermédiaire combine l'algorithme et l'architecture. Il permet la définition précoce de la plateforme d'implémentation et établit une continuité entre le modèle fonctionnel et le modèle architectural.
- En automatisant le passage entre le niveau système et le niveau architectural, dans le but d'accélérer la procédure de la conception des MPSoCs et de réduire la quantité des erreurs provoquées par le travail manuel dans un environnement unifié.
La pertinence de ce travail a été évaluée par son application à la conception du décodeur MP3 présenté dans ce mémoire.

MOTS CLES

Architecture multiprocesseur, monopuce, conception de haut niveau, conception systématique, synthèse matérielle/logicielle.

TITLE

Simulink-based MPSoC Design: Bridge between Algorithm and Architecture Design

ABSTRACT

The current fabrication technology allows the integration of a complex multiprocessor system on one silicon part (MPSoC for Multiprocessor System-one-Chip). A way to control the increasing complexity of these systems is to increase the abstraction level and to adopt the system level design. However, the increase of the abstraction level can make a huge gap between the system level concepts and those used for the hardware/software architecture implementation of MPSoC.
The objective of this thesis is to fill the gap between the two abstractions levels by proposing an efficient bridge between the algorithms development aid tools (Matlab\Simulink) and the architectures design tools (ROSES and macro-Cell builder). This is accomplished:
- By defining a transactional model in the Simulink environment. This intermediate model combines algorithm and architecture. It allows the early definition of the implementation platform and establishes continuity between the functional model and the architectural model.
- By automating the passage between the system level and the architectural level, to accelerate the MPSoCs design procedure and to reduce the errors quantity caused by manual design in a unified environment.
The relevance of this work was evaluated by its application to the MP3 decoder design presented in this memory.

INTITULE ET ADRESSE DU LABORATOIRE

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

ISBN : 978-2-84813-102