Méthode et outils de génération de code pour les plateformes multi-cœurs fondés sur la représentation de haut niveau des applications et des architectures

Amin El Mrabti

▶ **To cite this version:**

Amin El Mrabti. Méthode et outils de génération de code pour les plateformes multi-cœurs fondés sur la représentation de haut niveau des applications et des architectures. Informatique [cs]. Université de Grenoble, 2010. Français. NNT : . tel-00570692

HAL Id: tel-00570692

https://theses.hal.science/tel-00570692

Submitted on 28 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE GRENOBLE

N° attribué par la bibliothèque
978-2-84813-162-7

THÈSE

Pour obtenir le grade de

DOCTEUR de L'UNIVERSITÉ DE GRENOBLE

Délivré par l'Institut Polytechnique de Grenoble

Spécialité : Informatique

préparée au laboratoire **TIMA**

dans le cadre de l'école doctorale **«Mathématiques, Sciences et Technologies de l'Information, Informatique »**

Présentée et soutenue publiquement par

Amin EL MRABTI

Le 08 Décembre 2010

Titre

Méthode et outils de génération de code pour les plateformes multi-cœurs fondés sur la représentation de haut niveau des applications et des architectures

Jury

Mr. Ahmed JERRAYA,	Président,
Mr. Fabrice KORDON,	Rapporteur,
Mr. François TERRIER,	Rapporteur,
Mr. Emmanuel VAUMORIN,	Examinateur,
Mr. Frédéric PÉTROT,	Directeur de thèse,
Mr. Frédéric ROUSSEAU,	Co-directeur de thèse

Remerciements

Au terme de ce travail, je tiens à exprimer mes remerciements envers toutes les personnes qui ont contribué, à des degrés divers, au bon déroulement de ce travail.

Je remercie vivement mon directeur de thèse Mr Frédéric Pétrot, responsable du groupe SLS au laboratoire TIMA, pour sa disponibilité et ses conseils pertinents.

Je témoigne ma profonde reconnaissance à mon codirecteur de thèse Mr Frédéric Rousseau pour m'avoir guidé, suivi et aidé tout au long de l'élaboration de ce travail.

J'exprime également mes remerciements aux membres de jury : Monsieur Fabrice Kordon, Professeur à l'université de Pierre et Marie Curie, et Monsieur François Terrier, Professeur et responsable du laboratoire LISE au CEA LIST pour avoir accepté d'être rapporteurs de ce travail.

Je tiens à remercier Mr Ahmed Jerraya, directeur de recherche au CEA LETI, et Mr Emmanuel Vaumorin, Manager de projets stratégiques dans la société MDS, pour avoir accepté d'être examinateurs de ce travail.

Je tiens à saluer tous mes collègues doctorants, post doctorants et permanents du groupe SLS avec qui j'ai partagé cette expérience.

Résumé

La complexité des systèmes sur puce augmente pour supporter les nouvelles applications dans le domaine des télécommunications et du multimédia. La tendance actuelle des nouvelles architectures matérielles converge vers des plateformes multi-cœurs où les unités de calcul (processeurs, DSP, IP) sont interconnectées par un réseau sur puce, dont les interfaces réseau sont configurables. Pour ces nouvelles architectures, les environnements de génération de code binaire classiques ne sont plus adaptés.

Cette thèse propose un flot de génération de code de configuration pour le déploiement des applications de type flots de données sur les architectures à base d'IPs interconnectés à travers un réseau sur puce configurable. Ce flot est basé sur des modèles de haut niveau de l'application et de l'architecture et propose une méthodologie générique de partitionnement des ressources. Le processus de génération de code de configuration passe par plusieurs étapes modélisées par diverses représentations intermédiaires du système. La première est déduite depuis les modèles de haut niveau de l'application, de l'architecture et du partitionnement des ressources. Elle ne contient pas d'informations spécifique à la plateforme cible. La deuxième est générée par raffinement du premier modèle intermédiaire. Elle est spécifique à la plateforme cible et elle est exploitée par les outils de génération de code développés pour produire le code de configuration.

Le flot a été développé par la suite dans un environnement basé sur le standard IEEE 1685 (IP-XACT). Le flot proposé a été appliqué pour la génération et la validation du code de configuration en vue de déployer une application 3GPP-LTE de télécommunication sur la plateforme Magali du CEA LETI. Enfin, une généralisation pour intégrer ce flot dans un environnement classique de génération de code binaire exécutable par des processeurs, est proposée.

Abstract

The complexity of SoC is increasing to support new applications in the telecommunication and multimedia domains. The current trend of the new hardware architectures converges to multi-core platforms which gather several execution units (processors, DSP, IP) interconnected via a network on chip in which the network interfaces can be configured. For such architectures, the classical code generation tools and environments are no longer usable and suitable.

This thesis proposes a configuration code generation flow to deploy dataflow applications on IP-based architectures with configurable network on-chip. This flow starts with high level modeling of the application, the architecture and the resources partitioning methodology. The code generation flow goes through several phases which are modeled using various intermediate representations of the system.

The first intermediate representation is deducted from the high level models of the application, the architecture and the partitioning of the resources. This model does not contain any specific information related to the target platform. The second intermediate representation is generated through the refinement of the first intermediate model. It is specific to the target platform and is used by the code generation tools to produce the configuration code.

The code generation flow was developed in an environment based on IEEE 1685 (IP-XACT) standard. The proposed flow has been applied to generate and validate the configuration code to deploy the 3GPP-LTE application on the Magali telecommunication platform. Finally, a generalization of the proposed flow is proposed to integrate it in a classical generation flow of binary and executable code for processors.

Table des matières

Chapitre 1. Introduction......6
1.1. Contexte 6
1.2. Objectifs...... 6
1.3. Structure du document...... 8
Chapitre 2. Problématique 10
2.1. Contexte : Programmation des systèmes sur puce...... 11
2.2. Problème : Insuffisance des méthodes et outils classiques de génération de code logiciel...... 16
2.3. Problèmes liés à la modélisation des SoC pour la génération de code de configuration...... 20
2.4. Conclusion...... 22
Chapitre 3. Etat de l'art......23
3.1. Outils de génération de code de configuration 24
3.2. Inadéquation des méthodes et outils de génération de code logiciel exécutable par les processeurs...... 35
3.3. Conclusion...... 39
Chapitre 4. Définition d'un flot de génération de code configuration supportant les interfaces réseau configurables 40
4.1. Rappel de la problématique 41
4.2. Proposition d'une structure en couches du code de configuration...... 44
4.3. Définition d'un flot de génération de code de configuration...... 48
4.4. Modèle de représentation des plateformes à base d'IPs autour d'un réseau sur puce à interfaces réseau configurables...... 51
4.5. Modèle de représentation des applications orientées flot de données 54
4.6. Technique de partitionnement des ressources...... 57
4.7. Modèles intermédiaires et génération de code de configuration...... 59
4.8. Conclusion...... 62
Chapitre 5. Spécification du flot de génération de code de configuration dans un environnement IEEE 1685 (IP-XACT)......63
5.1. Rappel de la problématique 64
5.2. Présentation générale du standard IEEE 1685 (IP-XACT) 64
5.3. Spécification du flot de génération de code de configuration basé sur le flot IP-XACT 68
5.4. Extension du standard IP-XACT pour le modèle de représentation ARDL 71
5.5. Spécification du modèle de représentation de l'application APDL et des modèles de partitionnement...... 75
5.6. Automatisation du flot de génération de code de configuration 77
5.7. Intégration du flot de génération de code de configuration dans l'environnement Magillem 81
5.8. Conclusion...... 84

Chapitre 6. Génération de code de configuration pour le déploiement d'une application de télécommunication sur la plateforme Magali *86*
6.1. Modélisation de la plateforme Magali **87**
6.2. Présentation de l'application de télécommunication 3GPP-LTE **95**
6.3. Partitionnement de l'application 3GPP-LTE sur la plateforme Magali **98**
6.4. Génération des modèles intermédiaires **101**
6.5. Génération du code de configuration pour le déploiement de l'application 3GPP-LTE sur la plateforme Magali **107**
6.6. Simulation des fichiers de configuration **111**
6.7. Conclusion **112**
Chapitre 7. Vers un flot commun de génération de code *114*
7.1. Rappel de la problématique **115**
7.2. Présentation de l'environnement « APES » pour le développement du code logiciel exécutable par des processeurs **116**
7.3. Définition d'un flot commun de génération de code **118**
7.4. Modèle de représentation ARDL pour les architectures multiprocesseurs **118**
7.5. Description des flots de données KPN avec la représentation APDL **120**
7.6. Partitionnement des ressources **120**
7.7. Modèles intermédiaires **121**
7.8. Application du flot commun pour programmer les architectures multiprocesseurs **122**
7.9. Conclusion **125**
Chapitre 8. Conclusion et Perspectives *126*
8.1. Conclusion **126**
8.2. Perspectives **128**
Références bibliographiques *130*
Annexes *136*

Chapitre 1. Introduction

1.1. Contexte

Les systèmes intégrés sont de plus en plus présents dans notre vie quotidienne à travers les multiples fonctionnalités qu'ils fournissent (téléphonie, accès réseaux, appareils photo, GPS, musique MP3, vidéo, etc.). La conception de ces systèmes repose sur des plateformes matérielles, dont la complexité et les performances ne cessent de croître. La tendance actuelle converge vers des architectures matérielles multi-cœurs, c'est à dire contenant plusieurs unités de calcul (processeurs, accélérateurs matériels spécifiques, etc.) interconnectées par un réseau sur puce. Ces unités de calcul peuvent être de la même famille, on parle alors d'architecture homogène, ou de familles différentes, identifiées comme architecture hétérogène. De telles plateformes offrent des mécanismes assurant le parallélisme de calcul et de communication et sont conçues pour supporter des applications diverses. Un changement d'application implique de reprogrammer les unités de calcul et les supports de communications.

Ces nouvelles architectures multi-cœurs contribuent à supporter l'évolution rapide des normes applicatives liées aux marchés du multimédia et des télécommunications et à atteindre les performances de calcul et de communication requises. Mais la complexité de ces architectures rend très difficile le portage et le déploiement de telles applications. Des environnements logiciels d'aide au déploiement sont donc nécessaires pour aider les concepteurs systèmes.

Il existe des environnements de génération de code pour des architectures multiprocesseurs, ce qui facilite beaucoup le déploiement d'applications. Malheureusement, ces environnements se limitent à la production de code pour processeurs, et ne supportent pas à ce jour les récentes évolutions en termes de configuration (ou programmation) des supports de communication dans les réseaux intégrés sur puce (NoC : Network on Chip).

Au vu de ce contexte, cette thèse cherche à fournir de nouvelles méthodes et de nouveaux outils d'aide au déploiement d'applications sur des architectures matérielles multi-coeurs dont les supports de communication sont configurables.

1.2. Objectifs

Les techniques de développement d'une manière conjointe du logiciel embarqué et de la plateforme matérielle (co-développement ou co-design) ont pour but de produire un déploiement optimisé de l'application sur une architecture conçue sur mesure. Cela n'est plus viable pour des architectures matérielles complexes. Le coût élevé de telles solutions oblige les concepteurs à produire des plateformes matérielles adaptables à des familles d'applications.

Pour répondre à cette contrainte, le développement d'environnements (ou de flots) de génération de code devient une priorité. Les architectures matérielles, tout comme les applications, n'étant jamais vraiment figées, il faut que ces environnements prennent en entrée une représentation de l'application à déployer et une représentation de l'architecture matérielle incluant les informations nécessaires à sa programmation ou à sa configuration. Ces flots de génération sont généralement basés sur des modèles de haut niveau décrivant les applications, les plateformes cibles ainsi que le partitionnement des ressources. Une représentation de haut niveau réduit la complexité de la description d'entrée. Elle ne conserve que les informations utiles, et s'affranchit de détails inutiles à cette étape de conception du système.

Nous avons constaté depuis quelques années que les architectures sur lesquelles nous voulons déployer des applications complexes ne sont pas uniquement des architectures multiprocesseurs. En effet, certaines sont basées sur un ensemble de composants spécifiques (IP : Intellectual Property) connectés par un réseau sur puce (NoC). Une telle plateforme offre la possibilité de configurer les IPs et les composants de communication responsables de l'exécution et de l'ordonnancement du calcul au sein de l'IP. La génération de code pour ce type d'architecture devient alors une génération de fichiers de configuration pour la partie de calcul et la partie de communication. En effet, le code de configuration pour la partie de calcul est constitué d'un ensemble de paramètres qui initialisent les registres des IPs en vue de réaliser une tâche bien définie. Le code de configuration pour la partie communication correspond à la programmation d'une partie des réseaux sur puce (NI : Network Interface ou interface réseau) qui relient les IPs avec les routeurs du réseau sur puce. Pour orchestrer le tout à un niveau local à chaque IP, mais aussi au niveau global de l'application, d'autres fichiers de configuration et microprogrammes sont aussi à générer.

La production de ce type de code n'est pas encore supportée par les flots classiques de développement et de génération de logiciel embarqué. En effet, il s'agit là d'une problématique nouvelle. Un réseau sur puce est généralement construit et optimisé en fonction d'une application donnée. La famille de réseau sur puce, visée dans cette thèse, est générique et offre de nombreux services (ordonnancement des communications et du calcul), à travers la configuration de ses composants.

Dans un futur proche, nous prévoyons l'émergence d'architectures intégrant, d'un coté, des processeurs pour lesquels il faudra générer le code binaire exécutable, et d'un autre coté, des IPs reliés par un réseau sur puce configurable, pour lesquels il faudra générer également le code de configuration. Il nous faut donc réfléchir à un environnement commun pour la génération de ces deux types de code à partir d'une description de haut niveau de l'architecture, de l'application et du partitionnement des ressources.

L'objectif de cette thèse est de fournir des méthodes et des outils pour la génération du code de configuration afin de déployer des applications orientées flots de données sur les plateformes à base d'IPs interconnectés à travers un réseau sur puce configurable. Nous

souhaitons également généraliser ces méthodes et outils pour supporter à la fois la génération du code de configuration et du code logiciel binaire exécutable par les processeurs.

Le flot de génération de code de configuration proposé dans cette thèse est constitué de deux parties principales. La partie « avant » repose sur un modèle intermédiaire dit général représentant l'application déployée sur l'architecture. Ce modèle intègre toute l'information nécessaire à la suite du processus de génération, mais il n'est pas spécifique à la plateforme visée, ni au type de code à générer. Il contient en effet des informations structurelles de l'architecture et de l'application, mais ne comprend pas d'information sur le jeu d'instructions des processeurs ou sur la syntaxe des fichiers de configuration.

La partie « arrière » de ce flot s'appuie sur un modèle intermédiaire dit spécifique. Ce modèle est obtenu à partir du modèle intermédiaire général auquel viennent s'ajouter toutes les informations spécifiques à la plateforme visée et au type de génération souhaité. On précisera par exemple la syntaxe des fichiers de configuration, mais aussi le type des composants à configurer (entrée, sortie, …).

Ce flot de génération de code prend en entrée une description de la plateforme matérielle, une description de l'application et une solution de déploiement de l'application sur l'architecture. La description de l'architecture est basée sur le standard IEEE 1685 (IP-XACT) étendu pour répondre à nos besoins. Pour les applications, nous avons choisi une représentation qui intègre les formalismes classiques (KPN (Khan Process Network), SDF (Synchronous Data Flow)), mais qui permet aussi la description de fonctions particulières au domaine des télécommunications. Il ne s'agit donc pas d'un nouveau formalisme, mais juste d'une représentation textuelle sans sémantique associée. Les propriétés des modèles initiaux restent vraies, et l'utilisation des fonctions particulières modifie bien évidemment ces mêmes propriétés. On propose aussi dans cette thèse une représentation pour le déploiement de l'application sur les ressources de l'architecture qui se fait en deux étapes : une pour indiquer les contraintes et les règles d'association des ressources applicatives et matérielles, l'autre pour donner le déploiement final.

Enfin, cette thèse essaie de faire le lien entre un flot de génération de code binaire pour les processeurs et celui de génération de code de configuration. Il faut pour cela faire apparaître clairement les similitudes et les différences entre ces deux types de flots, et montrer qu'ils peuvent partager des étapes communes ou manipuler des modèles communs. L'objectif ici est d'anticiper sur les architectures matérielles du futur intégrant à la fois des processeurs et des IPs autour d'un réseau sur puce configurable.

1.3. Structure du document

Outre cette introduction générale, ce mémoire comporte sept chapitres. Le chapitre suivant détaille le contexte de ce travail et expose la problématique liée à la génération automatique du code de configuration, qui constitue le thème central de ce travail.

Le chapitre 3 présente et discute des travaux qui ont abordé cette problématique. Nous citons principalement des outils de configuration de réseaux sur puce pendant la phase de

conception, ainsi que certaines méthodes et outils de génération de code pour les architectures multiprocesseurs.

Le chapitre 4 présente l'approche que nous avons mise en œuvre pour la génération du code de configuration. Une proposition est également établie pour structurer en couches ce type de code.

Le chapitre 5 détaille comment nous avons spécifié et implanté cette approche dans un environnement basé sur le standard IEEE 1685 (IP-XACT).

L'application de cette solution est abordée au chapitre 6 à travers une étude de cas dans laquelle nous générons le code de configuration, qui permet de déployer une application de télécommunication 3GPP-LTE sur la plateforme configurable et largement paramétrable « Magali » [Lat08]. Dans ce chapitre d'expérimentation, nous mettons en évidence l'efficacité et les limitations des méthodes et outils que nous avons mis en œuvre.

En résumé, les chapitres 4 et 5 abordent les principales contributions de cette thèse et le chapitre 6 détaille l'expérimentation réalisée et les résultats obtenus.

Dans le chapitre 7, nous présentons notre dernière contribution. Elle consiste à généraliser le flot de génération de code pour viser à la fois le code de configuration et le code binaire exécutable par les processeurs. Nous étudions la possibilité d'utiliser les mêmes représentations de haut niveau pour la description de l'application, de l'architecture et du partitionnement des ressources.

Enfin, Le chapitre 8 expose les conclusions de ce travail et propose quelques perspectives potentielles.

Chapitre 2. Problématique

2.1. Contexte : Programmation des systèmes sur puce ... 11
2.1.1. Architecture des systèmes sur puce ... 11
2.1.2. Programmation des plateformes à base d'IPs interconnectés à travers un NoC ... 14
2.1.3. Exemple : Déploiement d'applications sur la plateforme Magali ... 14
2.2. Problème : Insuffisance des méthodes et outils classiques de génération de code logiciel. 16
2.2.1. Programmation des sous systèmes logiciels ... 16
2.2.2. Flot de génération de code pour les sous systèmes logiciels : possibilités et limites ... 18
2.2.3. Conclusion : Besoin de nouvelles solutions pour la génération de code de configuration pour les interfaces réseau configurables ... 20
2.3. Problèmes liés à la modélisation des SoC pour la génération de code de configuration.... 20
2.3.1. Inadaptation des modèles de calcul classiques à certaines applications orientées flots de données ... 20
2.3.2. Inadéquation des modèles intermédiaires ... 21
2.3.3. Difficultés de raffinement des modèles de haut niveau vers le code final à déployer ... 21
2.3.4. Difficultés d'automatisation des flots de génération de code de configuration ... 22
2.4. Conclusion ... 22

2.1. Contexte : Programmation des systèmes sur puce

Le progrès constant dans les technologies de l'électronique a favorisé l'apparition de systèmes sur puce (SoC pour System on Chip) de plus en plus performants qui embarquent de nombreuses fonctionnalités sur un unique substrat de silicium. Des appareils tels que les téléphones portables qui fournissent plusieurs fonctions (téléphone, accès réseaux, appareil photo, GPS, musique MP3, vidéo, etc.) témoignent de l'évolution croissante des systèmes sur puce. Ils sont également présents dans l'automobile (réglage des phares, contrôle du moteur, etc.), les téléviseurs HD, les « set-up box», etc. A la différence des ordinateurs, ils possèdent généralement des interfaces avec le monde extérieur (radio) et ont des contraintes de coût et de performance importantes.

Dans cette section, nous introduisons, dans une première étape, l'architecture d'un système sur puce. Nous décrivons, par la suite, comment déployer des applications sur ces systèmes, en particulier ceux à base de NoC à interfaces réseau configurables. Nous présentons, enfin, un exemple de ces architectures qui est la plateforme Magali [Lat08] du CEA/LETI. Nous nous intéressons à cette plateforme puisque le CEA/LETI nous a fourni son environnement de simulation dans le cadre d'un projet de recherche ANR [Hospi].

2.1.1. Architecture des systèmes sur puce

L'architecture générale d'un système sur puce est présentée dans la figure 2.1. Elle est constituée de composants de calcul (processeurs, IPs, DSPs (Digital Signal Processor)), d'un ensemble de périphériques et de composants de mémorisation, le tout interconnecté par un réseau de communication. On peut distinguer des composants de calcul programmables et non programmables. Les premiers sont généralement les processeurs. Ils exécutent des tâches applicatives diverses. Les seconds font référence aux composants IPs. Ils fournissent des fonctionnalités très spécifiques telles que la transformée de Fourier (FFT) ou la transformée de Fourier inverse (iFFT). Ils ont généralement des performances supérieures à celles d'un processeur. Certains composants non programmables proposent plusieurs fonctions spécifiques pouvant être sélectionnées durant une phase de configuration ; et ce, en écrivant certaines valeurs dans des registres de configuration.

Les composants périphériques étendent les possibilités du système en apportant des fonctionnalités d'accès à la mémoire (accès direct ou via le processeur), de gestion d'interruptions, de gestion de synchronisation, etc.

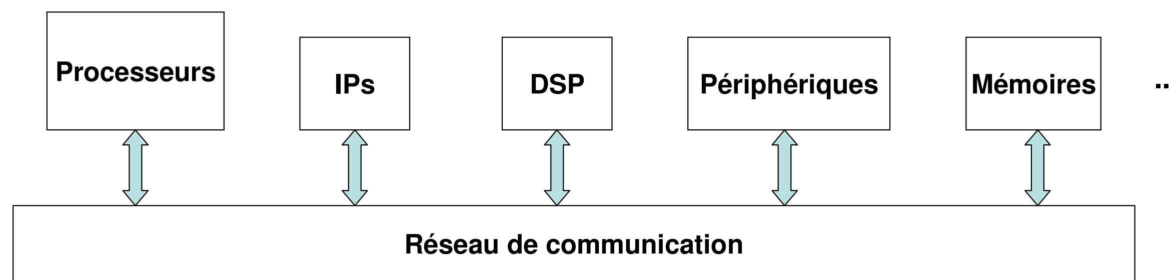


Figure 2.1. Architecture des systèmes sur puce

Tous ces composants communiquent par l'intermédiaire d'un support de communication qui peut être un bus ou un réseau sur puce. Le bus partage les communications entre tous les composants [Mur05]. Un bus hiérarchique, comme illustré dans la figure 2.2, est composé de plusieurs bus partagés, interconnectés par l'intermédiaire de ponts (« bridge »). Les infrastructures de communication classiques se basent sur les bus partagés. Or, ceux-ci ont montré leurs limites face à l'augmentation croissante des transferts de données entre les composants matériels de l'architecture, exigée par les applications. En effet, les bus partagés ne supportent pas des débits élevés et offrent peu de flexibilité quant au nombre de composants de calcul à interconnecter.

Face aux limitations des bus dans les systèmes sur puce, l'orientation actuelle des infrastructures de communication converge vers les réseaux sur puce, qui sont inspirés des réseaux informatiques classiques. Les NoCs supportent l'augmentation croissante de la complexité des applications en termes d'échanges de données. Ils permettent ainsi d'interconnecter plusieurs composants de calcul. Grâce aux multiples chemins de communication entre les ressources du SoC, les NoCs garantissent des critères de performance comme la latence, le débit et la bande passante.

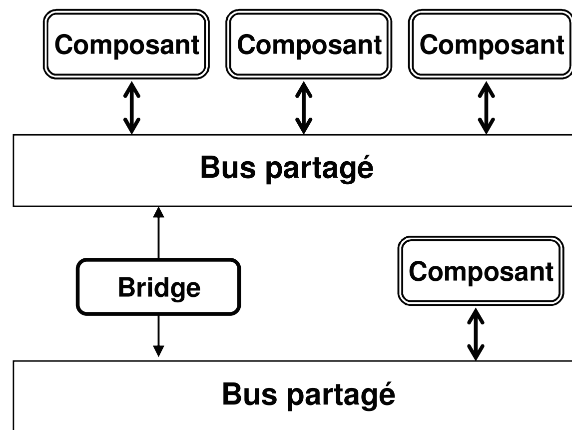


Figure 2.2. Bus hiérarchique

Les NoCs (Figure.2.3) sont constitués d'un ensemble de routeurs, auxquels s'interconnectent les composants via des interfaces réseau. Les routeurs fournissent des fonctions d'aiguillage et de routage des paquets de données échangés à travers la plateforme. Les interfaces réseau représentent les points d'entrée au NoC. Elles assurent l'acheminement des données de bout en bout. Ainsi, la communication entre deux composants distants connectés au NoC passe par deux interfaces réseau et plusieurs routeurs. Les interfaces réseau et les routeurs peuvent être configurables pour optimiser les chemins de communications des flux de données échangés à travers le NoC. Il existe plusieurs exemples de réseau sur puce : STNoc [Cop08], SPIN [Adr03] [Gue00], Xpipes [Ber04], Hermes [Fer04], FAUST [Cle05,Dur05], Æthereal [Goo05], etc.

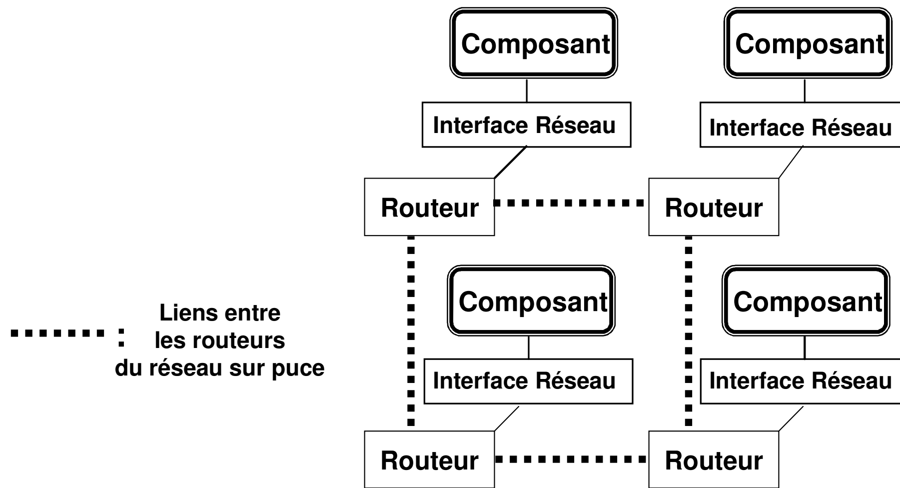


Figure 2.3. Réseau sur puce

Un système sur puce peut être vu et représenté comme un ensemble de sous systèmes logiciels et de sous systèmes matériels interconnectés par un réseau de communication (Figure.2.4).

Un sous système logiciel fait référence à l'exécution d'un programme par un ensemble de composants matériels. Il dispose lui même d'une partie matérielle et d'une partie logicielle. La partie matérielle comporte un ou plusieurs processeurs, des composants de mémorisation et des périphériques (TIMER, DMA (Direct Memory Access), périphériques d'entrée/sortie, etc.). La partie logicielle comporte tous les composants logiciels nécessaires à l'exécution de tâches à savoir le système d'exploitation, les pilotes des périphériques, le programme applicatif, les bibliothèques de communication, etc.

Un sous-système matériel fait référence à l'exécution d'une fonctionnalité spécifique par un composant de calcul dédié (IP). Les composants IPs sont des blocs matériels physiquement implantés sur la puce. En les configurant, nous pouvons choisir la fonctionnalité qu'ils doivent réaliser, et ce, en initialisant certains de leurs registres. A noter que chaque configuration identifie un comportement du sous système matériel.

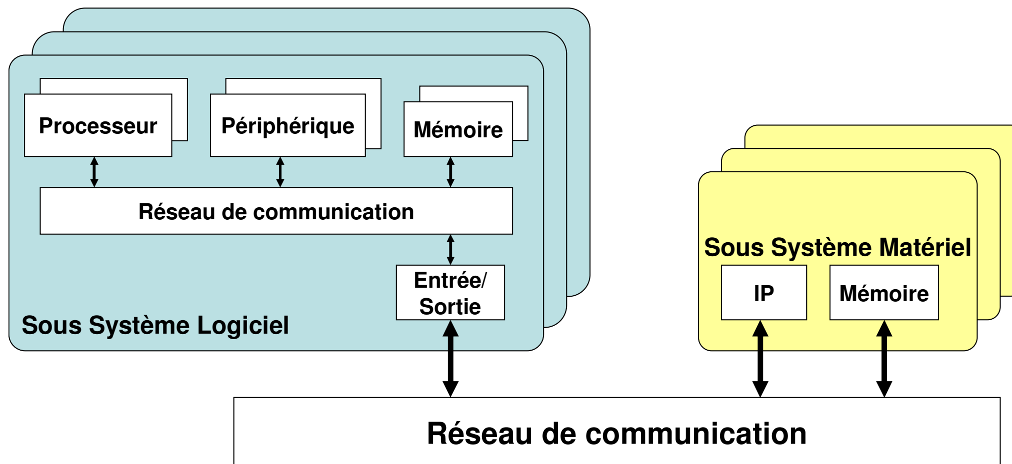


Figure 2.4. Sous système logiciel et sous système matériel

Nous focalisons les travaux de cette thèse sur le déploiement d'applications sur les sous systèmes matériels composés d'IPs et interconnectés à travers un réseau sur puce. Les plateformes que nous ciblons sont figées, c'est-à-dire que l'emplacement des IPs par rapport aux routeurs du réseau sur puce est fixe. Il s'agit de plateformes conçues pour cibler des familles d'applications précises connues à la conception. Nous présentons dans la section suivante la manière dont ces plateformes peuvent être programmées pour le déploiement d'applications.

2.1.2. Programmation des plateformes à base d'IPs interconnectés à travers un NoC

Le déploiement d'applications sur une plateforme figée commence par associer les tâches applicatives aux différents IPs. Un composant IP fournit une fonctionnalité à travers une configuration bien précise de ses registres. Plusieurs configurations peuvent être affectées à un seul IP s'il fournit plusieurs fonctionnalités. Les communications applicatives, représentant les échanges de données entre les blocs IP, sont assurées par le NoC, en particulier par les routeurs et les interfaces réseau.

Un mécanisme de contrôle et de synchronisation de l'exécution des tâches réparties sur les IPs doit être mis en place. Un processeur central peut jouer ce rôle et ordonnancer toutes les tâches et les communications. Cependant, un tel système centralisé a des limites vu le grand nombre d'interruptions à gérer, ce qui affecte les performances globales de la plateforme. Ainsi, le processeur central peut déléguer la gestion des communications et du comportement des IPs aux interfaces réseau réparties. Il se contente, quant à lui, de gérer les contrôles effectués par les interfaces réseau.

Ces interfaces réseau avancées qui fournissent des fonctionnalités de contrôle et de synchronisation des communications et des tâches sont configurables. En effet, elles ont besoin d'informations relatives aux échanges de données en entrée et en sortie de l'IP qui leur est connecté. Elles doivent disposer également des paramètres à associer aux registres de l'IP pour qu'il exécute la fonctionnalité demandée. Ces interfaces exécutent un jeu d'instructions simple qui permet d'exprimer le séquencement des tâches applicatives et des échanges de données, et définir ainsi le comportement de l'IP.

Magali est un exemple concret de plateforme qui dispose de ce genre d'interfaces réseau avancées. Nous présentons dans la section suivante les mécanismes de déploiement d'applications sur cette plateforme.

2.1.3. Exemple : Déploiement d'applications sur la plateforme Magali

La plateforme Magali est une plateforme dédiée à la réalisation des fonctionnalités des couches physique et MAC (Medium Access Control) pour des applications de télécommunication haut débit, de type 3G et suivantes (3GPP-LTE, WiMax, etc). Cette plateforme inclut un processeur unique ainsi qu'un ensemble d'IPs répartis sur le NoC. Le réseau sur puce est asynchrone. Il est constitué de routeurs à cinq ports reliés entre eux par des liens bidirectionnels, selon une grille à deux dimensions (2D-Mesh NoC). Cette plateforme, illustrée dans la figure 2.5, peut être représentée comme tout réseau sur puce qui interconnecte plusieurs IPs.

Pour chaque IP, plusieurs configurations élémentaires de calcul et de communication sont définies. Le code de configuration pour la partie de calcul est constitué par un ensemble de paramètres qui initialisent les registres des IPs en vue de réaliser une tâche bien définie. Le code de configuration pour la partie communication définit le nombre de données à transférer, le chemin qu'elles doivent suivre à travers le réseau ainsi que des informations concernant le contrôle de flux.

L'ordonnancement de ces configurations permet de définir le comportement de l'IP, à savoir les tâches à exécuter ainsi que les échanges de données à réaliser. Ainsi, un IP peut avoir un comportement différent selon l'ordonnancement choisi des configurations élémentaires. On appelle « ordonnanceur local » l'unité qui s'occupe de l'ordonnancement de ces configurations. Il est associé à l'interface réseau qui est connectée à chaque IP. Celle-ci définit le comportement des blocs matériels à travers un scénario d'ordonnancement des différentes configurations, écrit sous la forme d'un microprogramme.

Un « séquenceur global » associé à l'unique processeur de la plateforme permet d'ordonnancer les différents scénarii locaux (microprogrammes) en vue de définir le comportement global de la plateforme.

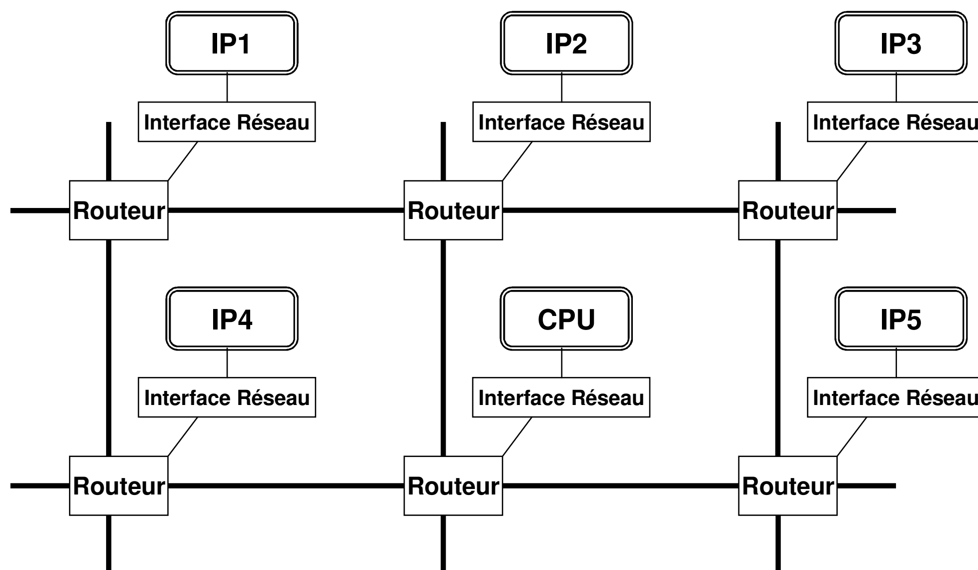


Figure 2.5. Plateforme à base d'IPs interconnectés par un réseau sur puce

Le code de configuration est exécuté au niveau des interfaces réseau configurables de la plateforme. En effet, l'interface réseau dans la plateforme Magali réalise plusieurs fonctionnalités telles que la configuration des unités de calcul (les IPs et l'unique processeur), l'ordonnancement des configurations de calcul et de communications locales ainsi que le contrôle de flux. Une présentation détaillée de l'architecture de l'interface réseau dans Magali, ainsi que son modèle de programmation se trouve dans le chapitre 3 de l'état de l'art.

Le développement du code de configuration pour cette architecture est complexe. Il est écrit, à l'heure actuelle, par la personne effectuant l'intégration de l'application sur la plateforme. Le code est constitué d'un grand nombre de fichiers contenant chacun plusieurs paramètres (5 fichiers de configuration par IP contenant les configurations de communication et de calcul). La grande difficulté dans la production de ce genre de code est de garantir la

cohérence des différentes valeurs, liées aux dépendances entre les divers paramètres dans les fichiers de configuration.

Nous nous intéressons dans cette thèse aux difficultés de génération de code de configuration pour déployer des applications sur des plateformes comme « Magali ». Nous nous focalisons principalement sur les interfaces réseau qui fournissent des services avancés d'ordonnancement et de synchronisation des communications et des tâches applicatives.

Nous exposons dans la suite les raisons pour lesquelles les méthodes et les outils de génération de code existants, pour les processeurs, ne sont pas applicables aux plateformes que nous ciblons.

2.2. Problème : Insuffisance des méthodes et outils classiques de génération de code logiciel

Dans cette section, nous présentons la structure du code logiciel qui cible les processeurs et les caractéristiques des méthodes et des outils qui le génèrent. Ensuite, nous discutons les possibilités d'application de ces méthodes et outils pour programmer les interfaces réseau configurables.

2.2.1. Programmation des sous systèmes logiciels

La programmation d'un sous système logiciel nécessite le développement des tâches applicatives exécutées par les unités de calcul. Le code applicatif fait appel à des modèles de programmation pour la gestion des tâches et des transferts de données entre elles. Par exemple, dans le but de programmer des applications parallèles pour des plateformes multiprocesseurs à mémoire partagée, le programmeur a besoin d'utiliser des mécanismes de gestion et de synchronisation de tâches. La norme Posix [But97] est un standard proposant de tels mécanismes.

Des systèmes d'exploitation (multi tâches, temps réel, etc.) sont aussi indispensables pour la programmation des sous systèmes logiciels. Ils fournissent les mécanismes d'ordonnancement et de gestion de l'allocation du processeur entre les différentes tâches applicatives. Le développement des systèmes d'exploitation nécessite le développement de primitives d'accès et de gestion des composants matériels de la plateforme. Des pilotes sont ainsi indispensables pour effectuer cette gestion, en particulier pour les composants périphériques.

Certaines plateformes fournissent une API (Application Programming Interface) sous forme de SDK (Software Developement Kit) à partir de laquelle l'utilisateur peut développer des applications. Cette API représente une interface d'accès à une couche logicielle contenant un intergiciel (« middleware »). Cette couche « middleware » est dépendante de la plateforme cible et du système d'exploitation. Nous pouvons citer «Android », « SymbianOS » et « Windows Mobile » pour les terminaux mobiles, qui fournissent des SDK facilitant le développement d'applications. Un émulateur de la plateforme et du système d'exploitation est généralement fourni pour valider les applications développées. Cette couche « middleware » joue le rôle d'intermédiaire entre le code de l'application d'une part et le transfert des données et leurs traitements sur la plateforme d'autre part.

Formaliser la programmation des sous systèmes logiciels en un ensemble de couches simplifie son développement. Nous pouvons ainsi structurer tous les composants logiciels, nécessaires à la programmation des sous systèmes logiciels. Cette structure en couches est appelée « pile logicielle » [Jer06] (Figure.2.6). Elle est composée de :

- La couche applicative qui est la couche de niveau supérieur. Il s'agit de la partie fonctionnelle du logiciel embarqué. Elle comporte une ou plusieurs tâches exécutées par le(s) processeur(s) du sous système logiciel. Le mode d'interaction et d'échange de données entre les différentes tâches suit un modèle de calcul. Un exemple très utilisé pour le développement des applications parallèles est KPN (Khan Process Network) [Lee95].

- Les couches « API de gestion des tâches » et « Gestion des tâches » fournissent des primitives pour la gestion des tâches applicatives (création, suspension, synchronisation des tâches, etc.). La première représente l'interface à l'ensemble des primitives de gestion de tâches implantées au niveau de la seconde couche.

- Les couches « API de communication » et « Communication » fournissent des primitives pour la gestion des communications entre les tâches décrites au niveau applicatif. La première représente une interface à l'ensemble des primitives de communication qui sont implantées au niveau de la seconde. Elles font appel à des fonctionnalités du système d'exploitation et de la couche « HAL API ».

  ⇨ Les couches de gestion des tâches et de communication constituent la couche « middleware » pour développer des applications contenant des tâches communicantes. Cette couche est extensible si nous voulons développer d'autres applications respectant des modèles de calcul différents.

- La couche du système d'exploitation englobe le système d'exploitation lui-même ainsi que les pilotes de périphériques.

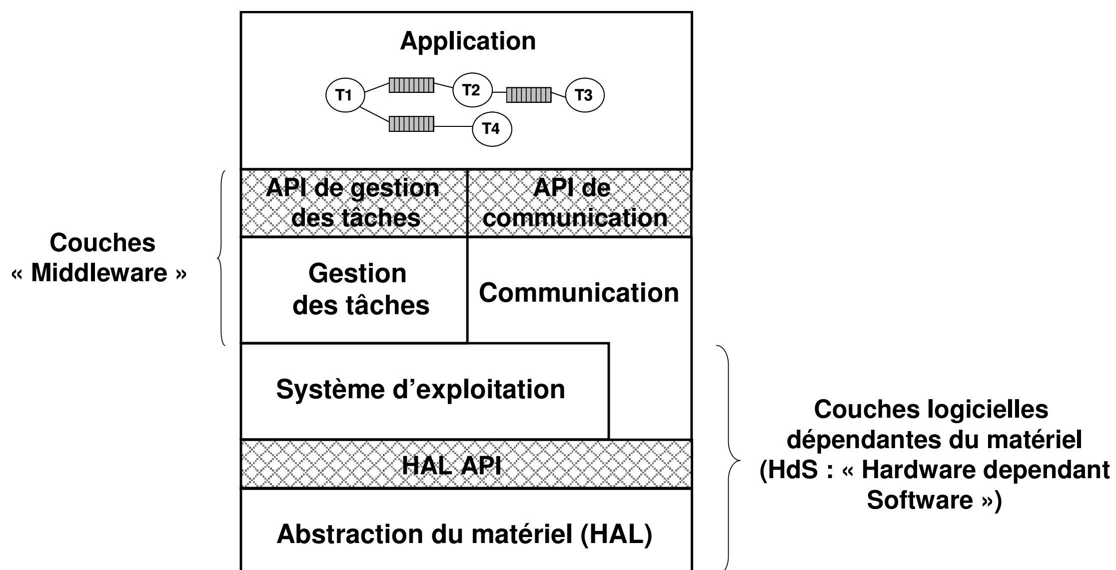


Figure 2.6. Modélisation du logiciel embarqué en pile logicielle

- Les couches HAL (Hardware Abstraction Layer) sont les couches d'abstraction du matériel. Elles comportent une interface générique pour les accès élémentaires aux composants matériels ainsi que l'implantation de cette interface, qui est spécifique à l'architecture cible. La couche « HAL API » est générique alors que son implantation (couche « HAL ») est spécifique au sous système logiciel et en particulier au processeur cible.

  Exemple : la fonction de changement de contexte « cxt_switch », dont la signature est « *void cxt_switch (cxt_type old_context, cxt_type new_context)* », représente un exemple de primitive générique de la couche «HAL API ». Cette fonction est développée de différentes manières, dans la couche « HAL » selon le type du processeur cible du sous système logiciel.

  ⇨ Les couches du système d'exploitation et HAL représentent le logiciel dépendant du matériel appelé HdS (Hardware dependant Software) dans [Dom09] et [Jer06]. La séparation du logiciel embarqué en deux parties -dépendante et indépendante du matériel- facilite son développement et améliore sa portabilité en favorisant sa réutilisation sur plusieurs plateformes.

Le paragraphe suivant présente les méthodes et les outils existants pour la génération du logiciel embarqué pour les sous systèmes logiciels. L'organisation de ces outils et la mise en œuvre des méthodes de déploiement introduisent le flot de génération de code logiciel.

2.2.2. Flot de génération de code pour les sous systèmes logiciels : possibilités et limites

Nous présentons dans ce paragraphe le flot de génération de code pour les sous systèmes logiciels. Il s'agit d'un flot connu [Gue07] et spécifique aux architectures multiprocesseurs (Figure.2.7). Le flot commence par la modélisation à haut niveau de l'application et de l'architecture et par la proposition d'un modèle de partitionnement des ressources applicatives sur les ressources architecturales, tel que défini dans [Thi07] et [Dsx]. Les applications généralement déployées sur des plateformes multiprocesseurs sont des applications orientées « flot de données » provenant de plusieurs domaines (multimédia, télécommunication, etc.). Il s'agit d'applications décrites par des tâches parallèles qui échangent des données via des canaux logiciels. Ces flots de données respectent généralement des modèles de calcul comme KPN et SDF [Lee87].

Le modèle de partitionnement des ressources applicatives sur des architectures multiprocesseurs consiste à associer les tâches applicatives aux processeurs et les canaux de communication applicatifs aux composants de mémorisation de la plateforme.

Le flot de génération de code utilise plusieurs outils et bibliothèques (systèmes d'exploitation, bibliothèques de communication, etc.). Certains outils peuvent générer le fichier « Makefile » pour la compilation du logiciel. Après compilation et édition de liens, un objet binaire est généré et exécuté par un processeur de l'architecture. Dans une architecture multiprocesseur, ce flot est potentiellement (sauf système SMP (Symmetric Multi Processing)) exécuté pour chaque processeur. Ainsi, un objet binaire est produit pour chacun

et sauvegardé dans ses espaces d'adressage. L'ensemble des outils et des bibliothèques peuvent être regroupés dans un environnement de développement logiciel.

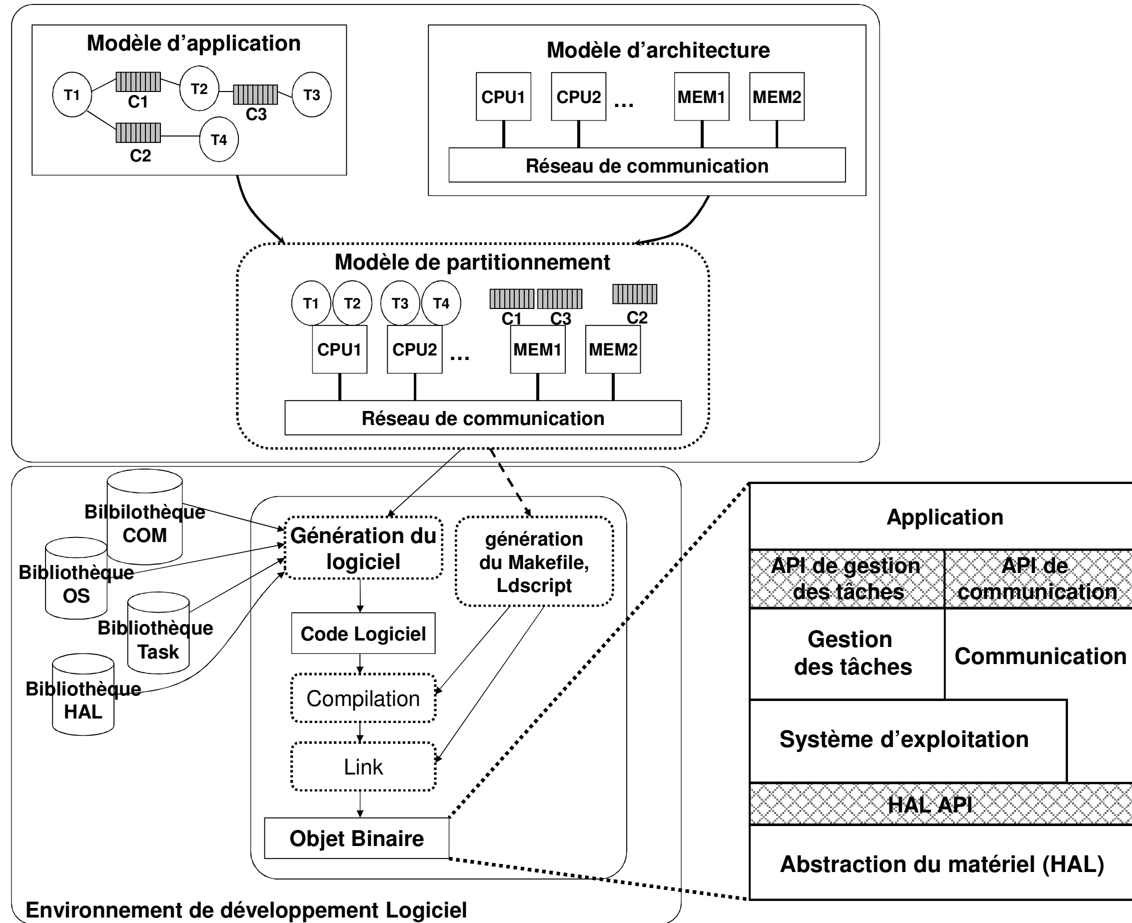


Figure 2.7. Flot classique de génération de code logiciel

Ce type de flot est spécifique au déploiement d'applications orientées flot de données, dont le modèle de calcul est KPN ou SDF, sur des architectures multiprocesseurs. Cette spécificité concerne la description de haut niveau des applications, des architectures et du partitionnement.

Ce flot ne peut pas être utilisé pour d'autres types d'architectures (architecture à base d'IPs interconnectés par un réseau sur puce) voulant exécuter d'autres types d'applications qui ne se représentent pas par les modèles de calcul KPN et SDF, i.e. comme le CSDF (Cyclo-Static DataFlow) [Bil95].

Il lui manque des modèles permettant de renprésenter l'aspect configurable de l'architecture. En plus, il n'existe pas d'outils capables de produire le code de déploiement sous la forme de configurations distribuées sur l'ensemble des interfaces réseau. Les plateformes, comme Magali, sont nouvelles et ne sont pas encore supportées par le flot classique de génération de code.

Il n'existe pas en notre connaissance, au moment où nous rédigeons cette thèse, de flots génériques de génération de code de configuration pour programmer les plateformes que nous ciblons.

2.2.3. Conclusion : Besoin de nouvelles solutions pour la génération de code de configuration pour les interfaces réseau configurables

A travers l'exemple de la plateforme Magali, nous avons montré que le code que nous voulons générer pour ce type d'architecture est un code de configuration. Le flot classique, présenté en figure 2.7, ne permet pas de le générer. En effet, le modèle de description de l'architecture ainsi que celui du partitionnement ne sont pas suffisants pour exprimer les aspects configurables des interfaces réseau et des IPs matériels dans une architecture telle que Magali.

L'évolution rapide de la technologie des réseaux sur puce, en particulier l'aspect configurable des interfaces réseau et leurs modes de programmation, fait ressentir le besoin d'améliorer les flots classiques de génération de code logiciel.

2.3. Problèmes liés à la modélisation des SoC pour la génération de code de configuration

La modélisation de haut niveau des SoC permet de fournir une vue qui facilite leur développement et réduit leur complexité. Cela nécessite la modélisation des applications, des architectures ainsi que leur partitionnement à un niveau d'abstraction approprié (TLM (Transaction Level Model [Systemc]) par exemple pour la modélisation des architectures). Plusieurs standards sont utilisés pour la modélisation de haut niveau comme UML (Unified Modeling Language), XML (eXtensible Markup Language), des bibliothèques C++ (SystemC [Systemc], SoCLib [Soclib]), etc. Cette modélisation représente une solution clé pour remédier au problème du retard que prend la production du code logiciel par rapport au saut technologique des architectures et des applications. Elle est généralement accompagnée par un ensemble de méthodes et d'outils de partitionnement et de génération de code. L'efficacité de la modélisation de haut niveau est conditionnée par l'efficacité des outils d'exploitation qui l'accompagnent (outils de transformation, de raffinement, de vérification, de génération, de simulation, etc.). Ces outils permettent le passage du niveau abstrait au niveau de réalisation en exploitant des informations additionnelles spécifiques à la plateforme cible. Ils sont généralement regroupés dans des environnements de développement logiciel.

Nous introduisons dans cette partie les problématiques liées à la modélisation de haut niveau des applications et des architectures ainsi qu'à la définition des représentations intermédiaires. Nous essayons, à travers la présentation de ces problèmes, de répondre aux besoins des flots de génération de code de configuration pour des plateformes ayant des interfaces configurables comme Magali.

2.3.1. Inadaptation des modèles de calcul classiques à certaines applications orientées flots de données

Les modèles de calcul connus pour la représentation des applications flots de donnés dans les domaines du multimédia et de la télécommunication sont principalement KPN et SDF. Les applications flots de données, dans ce cas, sont développées en utilisant des modèles de programmation parallèle connus (développement du modèle de processus de Khan à l'aide du standard *Posix* par exemple). Il se trouve que les applications ne cessent d'augmenter en

complexité au niveau de leurs algorithmes (algorithmes de traitement de signal et d'images par exemple). Un seul modèle de calcul devient insuffisant pour modéliser autant d'applications complexes et variées. Par exemple, le modèle de processus de Khan ne permet pas la représentation de l'arbitrage des transferts de données au niveau des canaux de communication. En effet, il ne représente pas la synchronisation des transferts de données entre les canaux de communication logiciels indépendants.

Or, l'arbitrage des communications est de plus en plus présent dans les applications récentes, en particulier dans le domaine des télécommunications. Le besoin se fait sentir de représenter ces informations d'arbitrage et de synchronisation dans les modèles applicatifs pour réaliser un partitionnement optimal des ressources.

2.3.2. Inadéquation des modèles intermédiaires

Dans les flots de génération de code, l'utilisation de modèles intermédiaires, qui regroupent les informations applicatives et architecturales, facilite l'extraction des données nécessaires aux outils de génération de code.

En effet, il existe plusieurs composants logiciels (tâches, primitives de système d'exploitation, primitives de communication, configurations, pilotes, etc.) qui sont exécutés par des composants matériels variés (DSP, ASIC, processeurs, infrastructure de communication, etc.). Le partitionnement ou le déploiement (mapping) de ces différents composants d'une manière optimale est un défit majeur dans la modélisation des systèmes sur puce. Il existe des modèles intermédiaires qui représentent l'affectation des tâches aux processeurs et l'association des canaux de communication logiciels aux composants de mémorisation [Chu08].

En revanche, lorsque l'on aborde des plateformes telles que Magali, ces modèles intermédiaires sont dans l'incapacité de représenter l'association des configurations ou des microprogrammes aux interfaces réseau configurables.

2.3.3. Difficultés de raffinement des modèles de haut niveau vers le code final à déployer

Dans les flots de génération de code, le passage depuis les modèles de haut niveau de l'application et de l'architecture vers le code final à exécuter nécessite généralement plusieurs étapes de raffinement. Les modèles de haut niveau ne contiennent pas assez d'informations pour la génération de code. Des détails concernant le partitionnement des espaces d'adressage, le système d'exploitation, les pilotes de périphériques, les adresses d'interruptions, etc., sont nécessaires à la génération de code final pour les processeurs. Ces informations supplémentaires peuvent être ajoutées par les développeurs à travers des environnements de développement logiciel.

Il s'avère que ces informations ne sont pas utiles pour des plateformes telles que Magali. En effet, le rôle classique du système d'exploitation et des bibliothèques de communication sont joués par des configurations dans la plateforme Magali.

Ainsi, nous cherchons dans cette thèse à identifier les informations adéquates à injecter aux outils de génération de code de configuration pour des plateformes à interfaces réseau

configurables. Le problème réside dans la définition de ces informations supplémentaires et dans leurs représentations dans des modèles intermédiaires.

2.3.4. Difficultés d'automatisation des flots de génération de code de configuration

Afin de garantir une génération de code rapide, les nouveaux flots de génération de code doivent être facilement automatisables. Il s'agit de définir un enchaînement d'exécutions de différents outils. Cet enchaînement est généralement le même pour une famille d'applications et de plateformes. Les principales difficultés dans l'automatisation des flots de génération de code dans ce contexte sont liés à :

(1) L'interopérabilité des outils de transformation et de génération de code ainsi que les types d'échanges qui peuvent se produire entre eux.

(2) L'expression d'un modèle d'automatisation du flot de génération de code avec les standards existants.

2.4. Conclusion

Nous avons présenté dans ce chapitre les architectures matérielles auxquelles nous nous intéressons dans cette thèse à savoir les systèmes à base de réseaux sur puce à interfaces réseau configurables comme la plateforme Magali. Nous avons mis l'accent sur l'importance des outils de modélisation et de génération de code pour supporter le saut technologique que connaissent les architecture multi-cœurs à base de réseau sur puce. Les flots de génération de code logiciel actuels sont spécifiques aux plateformes à base de processeurs et ne fournissent pas de fonctionnalités d'extension supportant de nouveaux types d'architectures.

Nous avons vu que le code à générer, pour une plateforme contenant des interfaces réseau configurables, comme Magali, est essentiellement un code de configuration.

Dans la première partie de cette thèse, nous essayons d'apporter des réponses aux questions suivantes :

1) Comment pouvons-nous formaliser le code de configuration, nécessaire à la programmation des interfaces réseau configurables, pour simplifier son développement ?
2) Comment pouvons-nous générer le code de configuration pour ces plateformes en se basant sur des modèles de haut niveau d'application, d'architecture et de partitionnement ?
3) Comment pouvons-nous automatiser le processus complet de génération de code de configuration ?

Nous souhaitons, dans la suite, aborder le problème de généricité des flots de génération de code. La principale question qui se pose à ce niveau est la suivante :

4) Pouvons-nous disposer d'un flot commun/général de génération de code qui pourra cibler plusieurs types de plateformes (plateformes à base de processeurs, plateformes à base d'IPs avec des interfaces réseau configurables, plateformes hétérogènes contenant à la fois les processeurs et les IPs, etc.) ?

Chapitre 3. Etat de l'art

3.1. Outils de génération de code de configuration 24
3.1.1. Architecture des réseaux sur puce 24
3.1.2. Les interfaces réseau classiques 25
3.1.3. Méthodes et outils de génération de code des réseaux sur puce 26
3.1.4. Interface réseau configurable du réseau sur puce Æthereal 27
3.1.5. Interface réseau configurable dans la plateforme Magali 30
3.1.6. Synthèse 35
3.2. Inadéquation des méthodes et outils de génération de code logiciel exécutable par les processeurs 35
3.2.1. Méthodes et outils de génération de code pour les processeurs 36
3.2.2. Synthèse 39
3.3. Conclusion 39

Ce chapitre présente l'état de l'art des flots de génération de code de configuration. Ce type de code est généré pour des architectures figées. Il s'agit de plateformes à base d'IPs dont l'emplacement est déjà prédéfini sur un réseau sur puce. Le code de configuration est constitué d'un ensemble de fichiers qui définissent le comportement des IPs et des interfaces réseau configurables de la plateforme. Le principal problème dans le développement de ce type de code se situe au niveau de la gestion de sa complexité (gestion d'un grand nombre de configurations et de leurs cohérences). C'est pour cela que nous nous intéressons aux moyens de génération du code de configuration d'une manière automatique à partir de la formalisation de celle ci.

Ainsi, l'état de l'art se focalise sur les méthodes et outils de génération de ce type de code pour les plateformes basées sur les réseaux sur puce et ayant des interfaces réseau configurables. Cette thématique a été rarement abordée dans les travaux de recherche existants. Les thèmes les plus traités se rapportant à la génération de code pour les réseaux sur puce sont autour de la synthèse d'architecture, ce qui n'est pas notre objectif.

Dans ce chapitre, nous introduisons d'abord les réseaux sur puces et les interfaces réseau classiques. Nous présentons ensuite les deux uniques exemples, qui existent à notre connaissance, traitant de la programmation des plateformes à travers de multiples configurations des interfaces réseau. Nous présentons par la suite les principaux flots classiques de génération de code logiciel. Nous exposons enfin les obstacles qui nous empêchent de les utiliser pour déployer des applications sur les plateformes que nous étudions dans le cadre de cette thèse. Nous rappelons que l'objectif est de réutiliser une même plateforme dans le déploiement de plusieurs applications appartenant à la même classe.

3.1. Outils de génération de code de configuration

3.1.1. Architecture des réseaux sur puce

Les réseaux sur puce [Gue00] [Ben05] sont des infrastructures de communication qui ont connu un grand essor durant ces dernières années. Ils sont de plus en plus utilisés dans les plateformes avancées en raison de la flexibilité et de la capacité à passer à l'échelle (scalability) qu'ils apportent dans les communications, en comparaison avec les systèmes à base de bus.

Le principe des réseaux sur puce s'inspire des réseaux entre ordinateurs classiques. La transmission de données entre diverses machines dans les réseaux classiques est définie selon le modèle en couches OSI (Open Systems Interconnection) [Zim80] de l' « *International Standard Organization* ». Il s'agit d'une architecture de référence en sept couches qui définit les étapes de traitement des informations entre le matériel et les applications. Pour concevoir et mettre en œuvre des réseaux sur puce, il a fallu adapter leurs caractéristiques au modèle OSI puisqu'il n'existe pas de modèle de référence pour la représentation des couches réseau propre aux réseaux sur puce. Ainsi, un réseau sur puce peut être représenté par les quatre couches basses du modèle OSI (Figure.3.1.b) [Bje06] [Ben01].

Les fonctionnalités des trois premières couches basses (« Physique », « Liaison de données » et « Réseau ») sont assurées par le composant routeur. D'une manière générale, le routeur est constitué de tampons en entrée et en sortie. Il réalise des mécanismes de routage et

d'arbitrage des données entre les différents nœuds. L'interface réseau joue le rôle d'adaptateur réseau. Elle fournit les fonctionnalités de la couche « Transport » du modèle OSI, à savoir la gestion des communications de bout en bout entre les nœuds connectés au réseau. L'interface réseau peut être configurable. Dans ce cas, elle fournit des fonctionnalités de contrôle de flux optimal [Rad04] ou d'ordonnancement et de synchronisation des tâches [Cle09]. Cette dernière fonctionnalité est généralement fournie en logiciel, à travers un système d'exploitation par exemple, dans les réseaux comme SPIN [Cha06] et STNoC [Cop08].

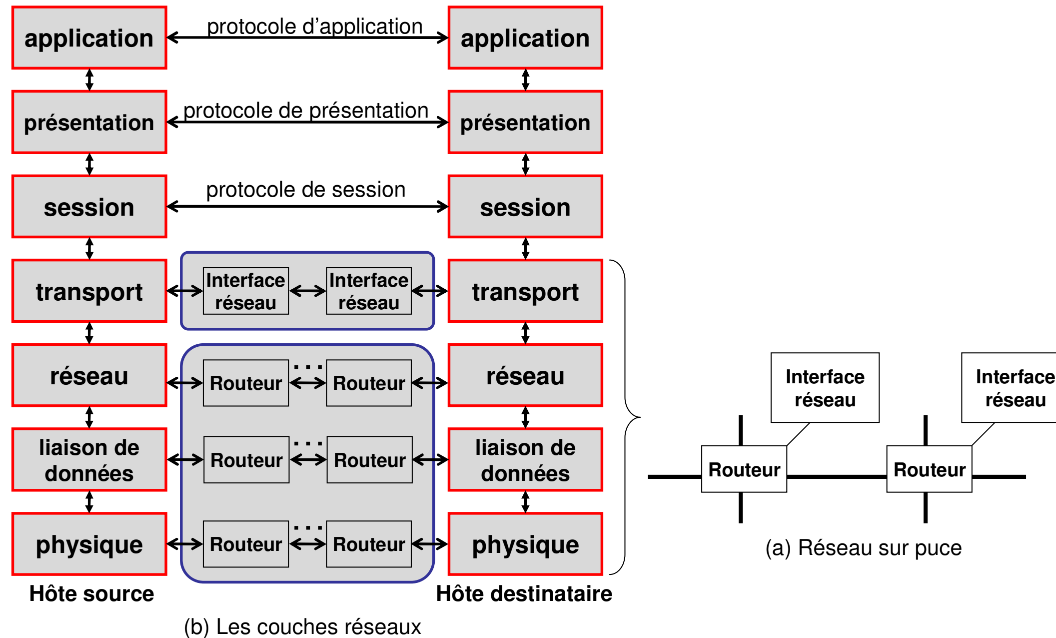


Figure 3.1. Les couches réseau pour un réseau sur puce

3.1.2. Les interfaces réseau classiques

L'interface réseau (Figure.3.2) est un composant qui relie les nœuds de la plateforme au réseau sur puce. Un nœud peut être un sous système logiciel ou un sous système matériel. L'interface réseau est interconnectée à un routeur d'un coté et au nœud de la plateforme de l'autre. L'ensemble des interfaces réseau constitue les points d'accès au réseau. Elles fournissent un ensemble de services qui masquent les mécanismes internes de transfert de données (mécanisme de routage, topologie du réseau, mécanisme de détection et correction d'erreur de transmission, contrôle de flux, etc.). Les interfaces réseau permettent la conversion des protocoles de communication physiques de chaque nœud vers ceux du réseau et vice versa.

Les interfaces réseau découpent les messages provenant de l'émetteur en paquets et ajoutent des informations supplémentaires relatives à la destination de ces paquets lors de l'envoi des données. De manière similaire, les interfaces réseau rassemblent les paquets reçus du réseau avant de transmettre le message complet au récepteur lors de la réception des données. L'interface réseau assure également le contrôle de flux des communications de bout en bout. Ce mécanisme permet d'éviter l'envoi de données vers une destination n'ayant pas d'espace suffisant dans ses tampons pour les recevoir. Les services mentionnés ci-dessus sont

fournis par les interfaces réseau classiques et sont indépendants des applications à déployer sur la plateforme.

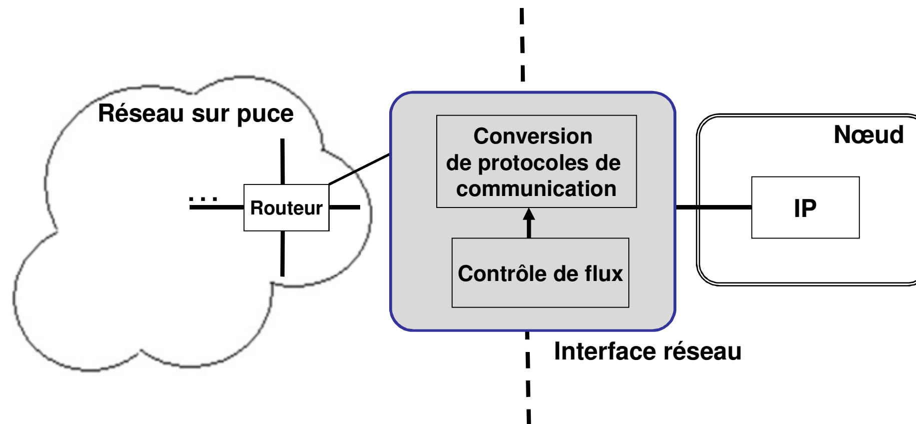


Figure 3.2. Fonctionnalités d'une interface réseau classique

3.1.3. Méthodes et outils de génération de code des réseaux sur puce

Dans la littérature, la majorité des outils développés pour les réseaux sur puce ciblent la synthèse d'architecture. Il s'agit d'outils et de générateurs qui permettent de concevoir une architecture de réseau optimisée pour une application ou une famille d'applications. Nous présentons brièvement les principaux travaux allant dans ce sens. Dans [Ber05], un flot de conception intitulé « Netchip » est proposé pour le réseau sur puce « Xpipes » [Ber04]. Le flot contient un outil pour la sélection de la topologie [Mur04], un outil pour une synthèse personnalisée de la topologie [Ati08] et un outil pour la génération du code matériel du réseau sur puce en SystemC [Jal04]. Un outil de synthèse du réseau µSPIDER [Eva06] est également présenté dans [Daf07]. Cet outil prend en considération des contraintes de latence et de bande passante.

Nous rappelons que nous ne nous intéressons pas à la synthèse d'architecture. Dans cette thèse, nous cherchons à développer des méthodes et des outils de génération de configurations pour les interfaces réseau avancées, afin de déployer des applications sur des plateformes déjà conçues. Ces dernières offrent des mécanismes de configuration dynamique des IPs et du réseau pour obtenir le comportement souhaité selon l'application à déployer.

Nous présentons dans la suite les deux uniques outils, à notre connaissance, qui permettent la génération du code de configuration pour deux NoCs dédiés. Le premier est lié au réseau Æthereal [Goo05]. Il permet de générer les configurations en prenant en considération des contraintes de latence et de bande passante. Le second est intitulé « ComC » et permet la génération du code de configuration pour la plateforme Magali. Nous avons choisi ces deux exemples puisque le code de configuration à générer est dépendant du flot de données des applications à déployer. Nous nous intéressons particulièrement à la plateforme Magali puisque nous allons l'adopter comme cas d'étude.

3.1.4. Interface réseau configurable du réseau sur puce Æthereal

Dans cette section, nous présentons le réseau sur puce Æthereal et l'architecture de ses interfaces réseau. Ensuite, nous détaillons comment ces interfaces sont configurées et nous détaillons les outils qui génèrent le code de configuration.

3.1.4.1. Présentation du réseau sur puce Æthereal

Æthereal est un réseau sur puce proposé par Philips. C'est une architecture qui assure une garantie de débit, de latence et de bande passante. Elle représente une infrastructure de communication sans topologie précise (Figure.3.3.a) pour les applications multimédia ou de télécommunication. La communication entre les ressources interconnectées au réseau est assurée par un mécanisme d'allocation de temps. Une unité de temps dans le réseau Æthereal est appelée « slot ». Une table d'attribution des « slot » aux différentes communications est configurée statiquement, au niveau des interfaces réseau, lors de la conception du NoC. Une configuration du réseau correspond à une application donnée. Une bande passante peut ainsi être garantie pour des communications en leur allouant un certain nombre de « slot ». La table d'attribution des « slot » peut être configurée d'une manière dynamique pendant l'exécution de l'application.

Le réseau Æthereal est configurable au niveau des interfaces réseau afin de réaliser les communications applicatives. Ces configurations permettent l'ordonnancement des communications mais n'interviennent pas au niveau de la partie de calcul de l'application.

3.1.4.2. Architecture des interfaces réseau dans Æthereal

L'interface réseau (Figure.3.3.b) dans le réseau Æthereal est le composant de base qui permet la configuration et la reconfiguration des communications à travers le réseau. Elle est composée d'une partie centrale nommée « NI Kernel » et d'une seconde partie nommée « NI Shell ». Le « NI Kernel » fournit les services de base d'une interface réseau classique, à savoir le contrôle de flux, l'assemblage et le désassemblage des messages reçus et émis. Les « NI Shell » sont des modules adaptateurs qui permettent la conversion des protocoles de communication entre le réseau et les IPs.

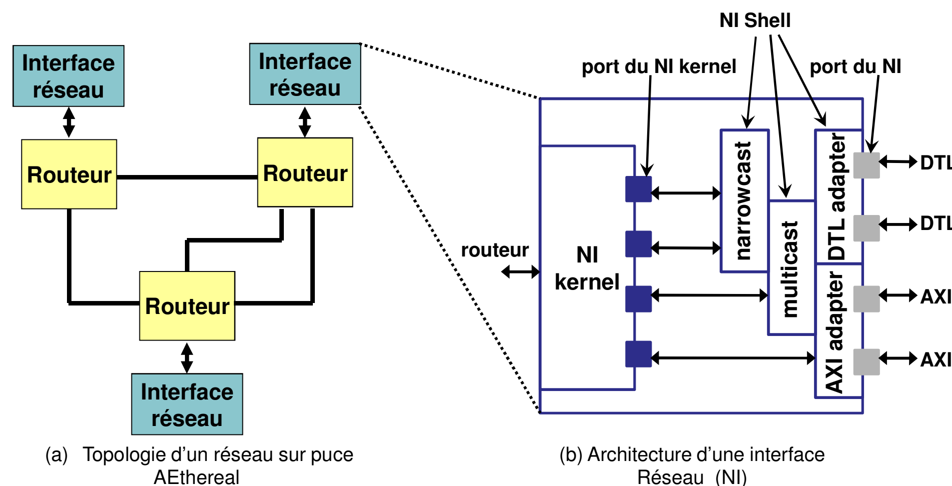


Figure 3.3. Réseau sur puce Æthereal [Rad04]

Ils supportent des services de diffusion de type « Norrwcast » et « Multicast ». Les protocoles de communication supportés par Æthereal sont DTL (Device Transaction Level), AXI (ARM) et OPC (Open Core Protocol). Le « NI Kernel » est relié aux adaptateurs « NI Shell » via des ports point à point.

Dans le « NI Kernel », il y a deux types de FIFO : les FIFO sources qui sont réservées aux messages à envoyer et les FIFO destinataires pour les messages à recevoir. La taille de ces zones mémoire est configurable pour l'optimisation des communications. Le mécanisme de contrôle de flux est assuré par un compteur de crédits associé à ces zones mémoires. Il est initialisé au moment du choix de la taille de la FIFO. Si une donnée est envoyée depuis une FIFO source, le compteur de crédits est décrémenté. Si une donnée est consommée par l'IP à partir des FIFO destinataires, le compteur de crédits associé à cette FIFO est alors incrémenté.

3.1.4.3. Mécanisme de configuration des interfaces réseau dans Æthereal

Æthereal permet de définir des connexions configurables et paramétrables entre les IPs qui communiquent, en garantissant un certain débit et une certaine latence. Les configurations des connexions se font à travers les interfaces réseau émettrices reliant les IPs. Elles sont dépendantes du flot de données applicatives et peuvent être activées ou désactivées au cours de l'exécution de l'application.

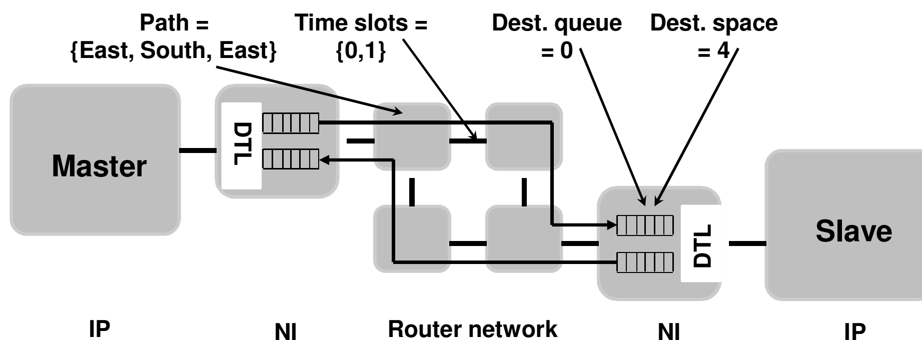


Figure 3.4. Configuration d'une communication entre deux IPs [Han07]

Comme illustré dans la figure 3.4, les informations indispensables pour configurer une interface réseau sont :

- Un chemin à travers le réseau pour le transfert des données (Elément « Path » dans la figure 3.4).
- Un identifiant de la FIFO destinataire pour la mise en correspondance entre la FIFO source de l'IP émetteur et la FIFO destinataire de l'IP récepteur.
- La taille de la FIFO destinataire qui reçoit les données. Cette information est indispensable pour fournir le service de contrôle de flux (« Dest.queue »/ « Dest.space » dans la figure 3.4).
- Un temps pour la communication spécifié par l'allocation d'un nombre de « slot ». Cette information est importante pour identifier les instants où la communication entre les ressources utilise les liens du réseau (« Time slots » dans la figure 3.4).

La définition des communications entre les IPs dans le réseau se fait par la configuration des registres des interfaces réseau. Cela est réalisé par un « mapping » mémoire appliqué aux ports de configuration des interfaces réseau. Ces ports offrent un accès aux registres des interfaces par le biais de transactions de lecture et d'écriture. Il existe des registres de configuration spécifiques au « NI kernel » et d'autres spécifiques aux « NI Shell ». L'ensemble de ces registres configurables au niveau des interfaces réseaux (Figure. 3.5), est classé en trois principales catégories:

- Les registres « path, queue, BE (Best Effort)/GS (Garanteeted Service), limit and space» : L'acheminement des données à travers le réseau est défini dans le registre « path » associé à l'IP émetteur. La FIFO de l'interface réseau destinataire vers laquelle les données sont transmises est définie dans le registre « queue ». Le registre « BE/GS » définit le choix des contraintes de qualité de service des communications. Enfin, le compteur de contrôle de flux, utilisé pour suivre l'occupation de la file d'attente de l'interface réseau destinataire, est défini dans le registre « space ».

- La table des « slot » : Ces registres participent à orchestrer d'une manière temporelle l'utilisation des chemins de communication et décident des moments d'envoi des données vers le réseau.

- Les registres du « NI Shell » : ces registres sont optionnels et permettent de configurer l'interface réseau pour supporter les protocoles de communication des composants IPs (DTL, AXI, etc.).

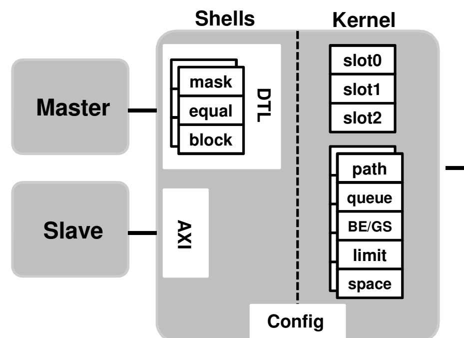


Figure 3.5. Liste des registres configurables dans une interface réseau [Han07]

3.1.4.4. Flot de génération de code du réseau Æthereal

Le flot de génération de code pour le réseau Æthereal, illustré dans la figure 3.6, est présenté dans [GDGPRR05].

Il prend en entrée un modèle de flot de données décrivant l'application (communication.xml), l'ensemble des IPs à interconnecter au réseau (ip.xml) ainsi qu'un modèle de contraintes (contraintes sur la bande passante et la latence des communications (constraints.xml)) en vue de générer la topologie du NoC. Il s'agit d'associer les IPs à leurs emplacements sur le réseau. Ensuite, le flot génère les configurations de communication pour les interfaces réseau (configuration.xml). Il produit enfin le code VHDL de la plateforme.

Si on dispose d'une nouvelle application flot de données à déployer, le flot exige une nouvelle génération du NoC (disposition des IPs par rapport aux routeurs). La génération des configurations est réalisée après que la plateforme soit définie, c'est-à-dire , après que les IPs à utiliser soient identifiés et placés sur le réseau.

A ce stade, nous avons présenté un exemple d'une interface réseau configurable qui fournit des services de communication pour une plateforme Æthereal à base d'IPs. L'interface réseau est configuratble à travers ses registres. Ces derniers contiennent des informations de communication de bas niveau se rapportant aux chemins de données, aux FIFO des interfaces réseau, aux adaptateurs avec les IPs, etc. L'outil de génération de ces configurations est spécifique au réseau Æthereal. Nous cherchons dans cette thèse à fournir une approche générale qui pourrait s'interfacer avec différents types d'interfaces réseau avancées.

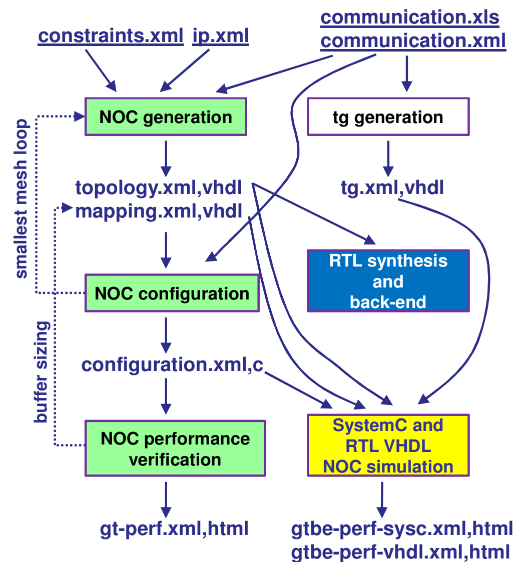


Figure 3.6. Flot de conception du réseau sur puce Æthereal [GDGPRR05]

Dans le prochain paragraphe, nous présentons un second exemple d'interface réseau configurable au sein de la plateforme Magali. Nous présentons également un outil de génération de code de configuration qui lui est spécifique.

3.1.5. Interface réseau configurable dans la plateforme Magali

On s'intéresse dans ce paragraphe à la plateforme Magali et à ses interfaces réseau configurables. Celles-ci permettent, contrairement aux interfaces réseau dans Æthereal, la gestion et l'ordonnancement à la fois de la partie communication et calcul de l'application à déployer.

3.1.5.1. Structure de la plateforme Magali

La plateforme Magali est une architecture parallèle semi hétérogène (Figure.3.7). Elle contient des IPs configurables fournissant des fonctionnalités spécifiques (IPs OFDM, les IPs de codage/décodage LDPC et Turbocodes, etc.).

Elle contient également des composants SME (Smart Memory Engine) qui sont des composants de mémorisation possédant les fonctionnalités d'un DMA programmable. La plateforme comporte des DSP basse consommation, appelés Mephisto, indispensables pour les applications de télécommunication. Ces DSP sont considérés comme des IPs car le code qu'ils exécutent est toujours fourni avec eux et figé. Enfin, la plateforme dispose d'un unique processeur central ARM11 pour le contrôle général de la plateforme. Les différents IPs sont interconnectés au NoC via les interfaces réseau, reliées elles mêmes aux routeurs.

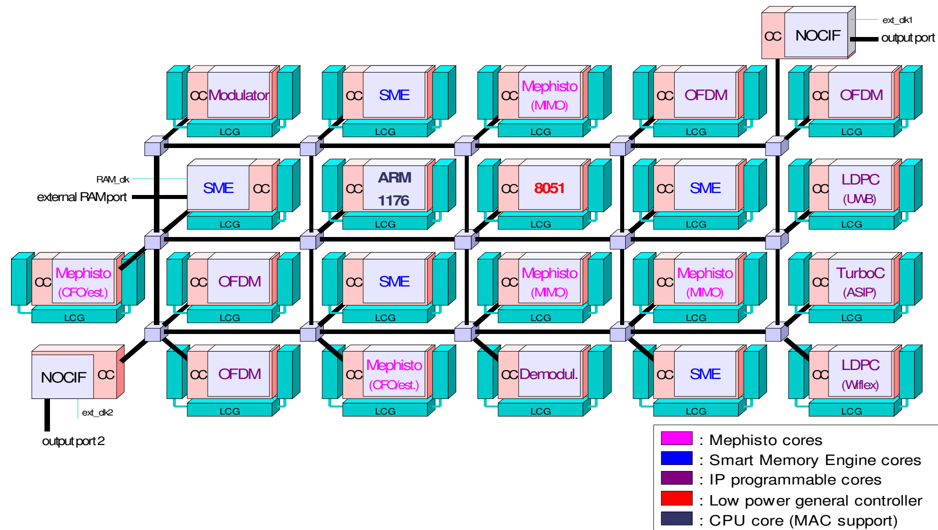


Figure 3.7. Architecture de la plateforme MAGALI

Chacun de ces derniers dispose de cinq ports : quatre pour la liaison avec d'autres routeurs et un pour interconnecter un IP ou le processeur central. Cette structure offre une grande flexibilité de routage pour échanger les données entre les différents IPs de la plateforme.

3.1.5.2. Interface réseau configurable : gestionnaire de configuration et de communication

Dans la plateforme Magali, chaque IP est relié à un routeur par l'intermédiaire d'une interface réseau avancée qui contient un bloc appelé CCC (*Communication and Configuration Controller*) [CLTV09]. Il s'agit du gestionnaire de configuration et de communication (Figure.3.8).

Ce bloc est le contrôleur global des communications et des traitements dans le réseau. C'est lui qui exécute l'ensemble des configurations. Chaque bloc CCC peut gérer jusqu'à quatre contrôleurs de communication en entrée (ICC : Input Configuration Controller) et quatre contrôleurs de communication en sortie (OCC : Output Configuration Controller). Le nombre de « ICC » et « OCC » à utiliser dépend du type de l'IP auquel le CCC est interconnecté. Il fournit différentes fonctionnalités, à savoir la configuration du processeur central et des différents IPs au niveau des communications et des traitements. Il fournit un séquenceur local des configurations (*local scheduling*) et un mécanisme de contrôle de flux des données.

Le séquenceur local de configurations permet de micro-programmer et d'exécuter localement un enchaînement de configurations pour les contrôleurs de communication (ICC, OCC) et pour l'IP. Il utilise à ce niveau un ensemble de primitives simples telles que RC pour « Request Configuration » auxquelles nous associons un identifiant de la configuration à exécuter. Il existe également des primitives d'ordonnancement des configurations qui permettent de décrire une boucle locale (LL pour Local Loop) et une boucle globale (GL pour Global Loop).

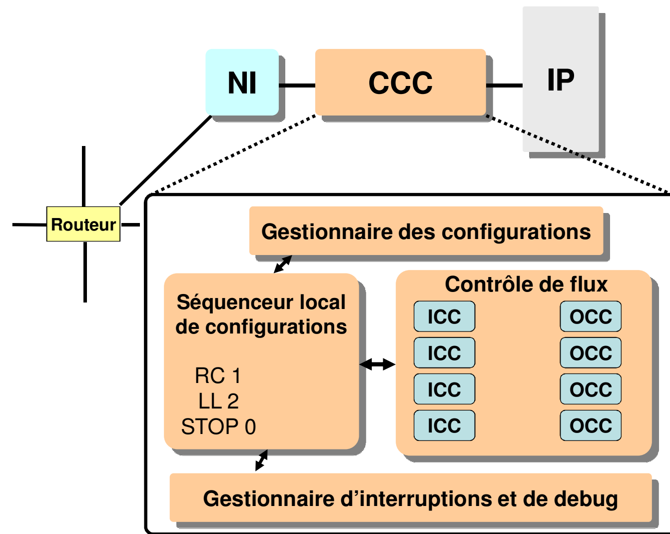


Figure 3.8. Architecture d'un gestionnaire de configuration et de communication

Lors de l'exécution d'une configuration, le séquenceur demande au gestionnaire des configurations si elle est chargée dans sa mémoire. En effet, le gestionnaire des configurations joue le rôle d'une mémoire cache du CCC, en sauvegardant les configurations couramment exécutées. Si la configuration n'est pas disponible chez le gestionnaire, ce dernier est capable de la charger et de la fournir au séquenceur local des configurations. Les configurations sont généralement sauvegardées dans une table située à l'extérieur du CCC, dans un composant SME (Smart Memory Engine).

Le contrôle de flux est lié aux contrôleurs de communication (ICC et OCC) et s'appuie sur un mécanisme de crédits : l'IP récepteur envoie un message particulier à l'IP émetteur pour lui indiquer le nombre de données qu'il peut recevoir. On garantit ainsi que les données émises ne seront ni perdues ni bloquées sur le réseau faute d'espace mémoire dans l'unité destinatrice.

3.1.5.3. Les fichiers de configuration pour le déploiement d'applications sur la plateforme Magali

Dans ce paragraphe, nous allons décrire les fichiers de configuration nécessaires pour le déploiement d'applications sur la plateforme Magali (Figure 3.9). Ces fichiers de configuration sont dans un format texte lisible et testable en simulation de la plateforme Magali.

Chaque nœud dispose d'un ensemble de fichiers de configuration:

1) Le fichier "core.cfg" contient des paramètres de l'IP spécifiant les fonctions qu'il peut fournir. Chacune dispose d'un identifiant et d'un ensemble de paramètres associés aux registres de l'IP. Il s'agit des configurations de calcul.
2) Le fichier « ip.str » contient des paramètres qui spécifient le nombre de contrôleurs ICC et OCC auxquels l'IP est connecté.

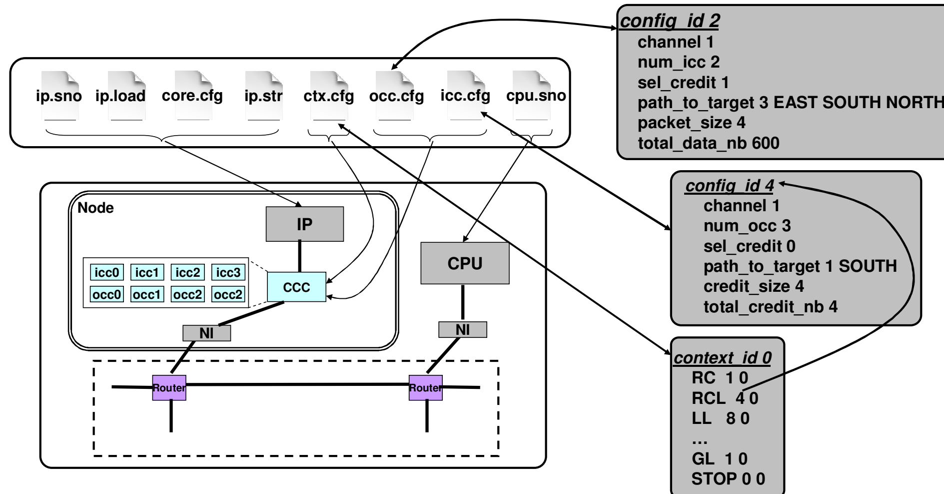


Figure 3.9. Ensemble de fichiers de configuration pour le déploiement d'applications sur la plateforme Magali

3) Le fichier « icc.cfg » contient un ensemble de configurations de communication. Chacune dispose d'un identifiant unique et définit la manière dont l'IP doit recevoir les données. Elle contient des informations relatives au nombre de données à recevoir, à l'identifiant du contrôleur de communication de type « OCC » qui les envoie, au chemin à suivre par les données à travers le réseau sur puce, ainsi qu'au compteur de crédits pour assurer le contrôle de flux.
4) Le fichier « occ.cfg » contient également un ensemble de configurations de communication. Chacune dispose d'un identifiant et indique la manière dont l'IP doit envoyer les données. Elle contient des informations relatives au nombre de données à envoyer, à l'identifiant du contrôleur de communication de type « ICC » qui va les recevoir, au chemin à suivre par les données à travers le réseau sur puce, ainsi qu'au compteur de crédits pour garantir le contrôle de flux.
5) Le fichier « ctx.cfg » référence les configurations de calcul définies dans le fichier "core.cfg", dans des microprogrammes, pour les orchestrer en vue de définir le comportement de l'IP. Il référence et ordonnance également les configurations de communications (définies dans les fichiers « icc.cfg » et « occ.cfg »). Afin d'exprimer l'ordonnancement, les primitives « RC: Request Configuration, LL: Local Loop, GL: Global Loop, etc. » sont utilisées dans les microprogrammes écrits dans les fichiers « ctx.cfg ».
6) Le fichier « ip.sno », référence les configurations et les microprogrammes définis dans le fichier « ctx.cfg » en vue de définir le comportement local de l'IP.

7) Le fichier « cpu.sno » développe le comportement global de l'application et est associé à l'unique processeur de la plateforme. Il contient des microprogrammes qui séquencent tous les types de configurations définies précédemment. Des primitives telles que (*LOAD_RES* <*ip_name*> <*ip_id*>) sont utilisées dans ce fichier. La configuration à charger est identifiée par le paramètre <*ip_id*> et doit correspondre à un paramètre nommé « source_id » du fichier « ip.sno ».

Comme on peut le constater à la lecture de la longue mais nécessaire énumération précédente, les fichiers de configuration pour le déploiement d'applications sur la plateforme Magali sont inter-dépendants et nombreux. Cette forte dépendance représente une réelle difficulté pour un déploiement manuel des applications. En plus du nombre important de fichiers de configuration, la modification d'un seul, en phase de débogage ou d'optimisation, entraîne la modification de certains autres. Pour ces raisons, il est nécessaire de disposer d'un outil de génération automatique des configurations.

Le CEA propose un outil de génération automatique des configurations (ComC) sous format binaire. Le code généré est associé au seul processeur de la plateforme et exécuté dans la plateforme réelle. Une fois généré, ce code ne peut être ni débuggé ni maintenu. Pour cela, un modèle de simulation de la plateforme Magali a été développé par le CEA. Ce dernier prend en entrée les configurations au format texte et permet de simuler et de visualiser les exécutions des tâches et des échanges des données entre les IPs. Ce mode de développement des configurations est réalisé manuellement. Ainsi, un outil de génération des configurations sous format texte est indispensable.

3.1.5.4. « ComC » : outil de génération de code de configuration binaire pour la plateforme Magali

« ComC » [Chw08] [Hospi] est un générateur de code de configuration au format binaire pour la plateforme réelle (Figure 3.10). Il ne génère pas les fichiers de configuration lisibles que nous voulons développer pour déployer une application sur le simulateur de la plateforme Magali. Il est écrit en Java, et prend en entrée des descriptions de la plateforme et de l'application dans un format XML. Le modèle applicatif est très spécifique à Magali. Il décrit les tâches comme étant les nœuds de la plateforme. Il associe à chacune un ensemble de configurations propres à la plateforme Magali. Les communications entre les tâches sont décrites explicitement à travers des liaisons entre les « ICC » et les « OCC ». Le modèle de la plateforme liste les IPs référencés au niveau du modèle applicatif. Il est à noter qu'il n'existe pas de modèle de partitionnement de ressources dans cet outil. Le modèle applicatif joue ce rôle.

Ces modèles sont ensuite compilés pour produire les fichiers suivants :

- Un programme C compilable pour le processeur ARM intégré au circuit, contenant le code de démarrage (boot) de l'ARM, le code d'initialisation matérielle de la plateforme Magali, ainsi que le code de configuration et de démarrage de l'application.
- Des fichiers binaires regroupant, pour chaque IP de la plateforme, les configurations de communication et de calcul nécessaires au déroulement de l'application. Ces configurations sont stockées à l'initialisation du système dans un ou plusieurs

composants SME. Chaque unité envoie au cours de son exécution des requêtes au SME pour demander les configurations qui lui sont nécessaires.

- Un fichier scénario servant à démarrer l'exécution de l'application. Il correspond à la phase de chargement des mémoires pour la plateforme réelle, qui peut se faire par exemple par une liaison Ethernet.

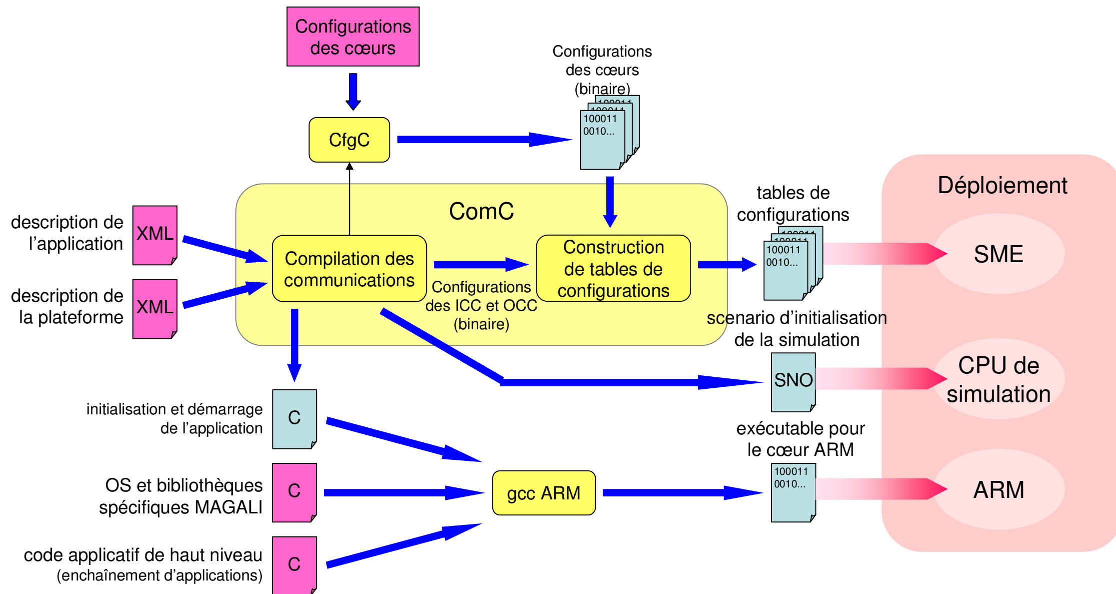


Figure 3.10. Utilisation de l'outil ComC pour déployer une application sur la plateforme Magali [Hospi]

Là encore, l'outil est totalement ad-hoc, et doit être développé d'une manière différente pour chaque plateforme. Il ne peut par exemple pas être utilisé sur Faust, la version précédente de Magali.

3.1.6. Synthèse

Nous avons vu dans cette première partie les deux exemples existants, à notre connaissance, d'interfaces réseau configurables dont les configurations sont dépendantes du flot de données applicatives (Magali, Æthereal). Pour chacune des deux plateformes, un outil spécifique a été développé pour générer le code de configuration. Nous cherchons dans cette thèse à fournir une approche commune de génération de code de configuration.

Nous discutons, dans la seconde partie de ce chapitre, de l'inadéquation des méthodes et outils de génération de code logiciel pour les architectures multiprocesseurs, aux types de plateformes et de code que nous ciblons.

3.2. Inadéquation des méthodes et outils de génération de code logiciel exécutable par les processeurs

Il existe plusieurs méthodes pour la génération du code logiciel exécutable par des processeurs, qui se basent sur la modélisation de haut niveau des architectures et des

applications. Ces représentations de haut niveau sont décrites en langage UML, en SystemC, en XML ou avec certains langages spécifiques tels que Simulink, etc. Le code logiciel généré, sous format binaire, inclut le code applicatif, le système d'exploitation, les librairies de communication et de gestion de tâches, les pilotes des périphériques, etc. Or, le type de code que nous souhaitons générer pour les interfaces réseau configurables est très différent du code exécutable par les processeurs.

En effet, il ne se base pas sur un système d'exploitation, l'ordonnancement des tâches étant effectué à travers des configurations. Cet ordonnancement des tâches et des communications est explicite dans notre cas alors qu'il est implicite quand on utilise un système d'exploitation. D'autre part, la mise en œuvre des communications dans le logiciel exécutable par des processeurs est souvent explicite à travers des bibliothèques de communication (primitive read(), write()). Dans notre cas, les communications sont décrites par des configurations contenant des informations applicatives (taille de données à transférer, par exemple) et d'architectures (chemin à prendre à travers le réseau pour transférer les données, par exemple).

3.2.1. Méthodes et outils de génération de code pour les processeurs

Les méthodes et les outils de génération du code logiciel exécutable par les processeurs intègrent des modèles de haut niveau et des générateurs. Ils sont toujours conçus et développés pour une famille d'applications et d'architectures bien déterminées. Cependant, il s'avère qu'ils ne répondent pas aux spécificités des plateformes que nous traitons dans cette thèse.

En effet, nous visons la génération de configurations en se basant sur une représentation des flots de données qui autorise la description de l'arbitrage des données d'une part, et ne comporte pas d'algorithme applicatif d'autre part. Or, dans les architectures multiprocesseurs, l'arbitrage des données n'est pas généralement exprimé puisqu'il est écrit dans le code des tâches et géré par les bibliothèques de communication. Dans notre cas, il s'agit de fournir une représentation qui interprète le comportement des communications applicatives.

Il n'est également pas nécessaire, dans les architectures que nous étudions, de représenter les algorithmes applicatifs puisqu'ils sont déjà implantés dans les IPs. On pourrait croire que le travail en est simplifié, mais il n'en est rien. En effet, pour les IP multi fonctions, il faut extraire leurs caractéristiques (registres, adresses, paramètres, etc.) qui permettent de les mettre dans l'état voulu pour qu'ils assurent les fonctionnalités souhaitées. Dans le code logiciel binaire classique, il s'agit de l'appel à une fonction (d'une bibliothèque) avec les paramètres correspondants.

En résumé, Les modèles utilisés pour générer le code logiciel pour les processeurs sont principalement des modèles comportementaux, avec une vue locale du comportement de chacun des processeurs et de l'application. Pour le reste, on suppose qu'il existe un support (de communication, de gestion des tâches, etc) issu d'une bibliothèque de fonctions. Dans notre cas, le comportement de l'application et de la plateforme sont définis en ordonnançant les configurations de communication et celles des IPs. Il faut là une connaissance globale des communications applicatives, des configurations des IPs et du partitionnement de toutes ces

configurations sur les interfaces réseau réparties sur la plateforme, ce qui n'est pas le cas dans un processus de génération de code logiciel classique.

Nous allons, dans ce qui suit, présenter les principaux outils existants de génération de code logiciel.

L'outil SynDEx [Rau06] (Synchronized Distributed Executives) proposé par l'INRIA, permet le déploiement des applications de traitement du signal sur des architectures hétérogènes dédiées. Cet outil applique une méthodologie formelle d'adéquation, appelée « AAA (Adéquation Algorithme Architecture) » [Nia04], entre les algorithmes applicatifs et les architectures. Un formalisme de graphe de données (Data Flow Diagram : DFG) est défini pour modéliser les différentes opérations de l'algorithme de l'application. Un formalisme pour la description des différents composants matériels est également défini.

De son coté, L'outil DSX (Design Space Explorer) [Dsx], proposé par le Lip6, développe du logiciel destiné aux applications multimédia pour des architectures multiprocesseurs. Les applications sont décrites sous forme d'un graphe de tâches s'exécutant en parallèle et communicant à travers des canaux de communication logiciels. Ces canaux de communication sont spécifiques à l'outil DSX et appelés MWMR (Multi Writer Multi Reader pour multi-écrivain multi-lecteur [Fau06]). Ils se comportent comme une FIFO logicielle à laquelle peuvent être connectées plusieurs tâches productrices et consommatrices de données, sans possibilité d'arbitrage. Les accès aux canaux sont protégés par des verrous. Un seul accès au canal est permis à un instant donnée, afin d'assurer la cohérence du contenu de la FIFO logicielle. Le partitionnement des ressources associe les composants applicatifs aux processeurs et aux segments mémoire des périphériques ou des composants de mémorisation.

L'environnement de développement logiciel/matériel Daedalus [Tho07] ne répond également pas à nos besoins. En effet, il permet la génération de code exécutable sur les processeurs à partir d'une application séquentielle initiale. Cette dernière est parallélisée dans un premier temps grâce à l'outil KPNgen [Ver07]. Le graphe des tâches obtenu, conforme au modèle de calcul KPN, est ensuite décrit à haut niveau au format XML. Un outil appelé SESAME [Nik08] est ensuite utilisé pour fournir une description de haut niveau de la plateforme et du partitionnement des ressources. Ces trois modèles sont ensuite explorés par l'outil ESPAM [Nik06] pour générer le code exécutable pour chaque processeur de la plateforme. Ce code final inclut le comportement des tâches et leurs synchronisations ainsi que le plan mémoire de tout le logiciel.

Dans le projet SHAPES (Scalable Software Hardware Architecture Platform for Embedded Systems) [Spo07], un flot de génération automatique du logiciel binaire pour les architectures multiprocesseurs est proposé. Il supporte les applications parallélisées dont le code source est déjà fourni. Les modèles de haut niveau sont fournis par ETH Zurich avec le langage DOL (Distributed Operation Layer) [Thi07]. Le premier permet de représenter les architectures multiprocesseurs en décrivant les composants de calcul et de communication. Le second décrit les applications en respectant le modèle de calcul KPN. Les différentes tâches applicatives communiquent à travers des canaux logiciels à une seule entrée et à une seule sortie. Le modèle de partitionnement permet d'affecter les tâches aux processeurs et les canaux de communication logiciels aux mémoires de l'architecture. Les informations applicatives et architecturales sont réunies dans un modèle intermédiaire unique au format SERIF (SERvice oriented Intermediate Format) [Chu08]. Il s'agit d'un format intermédiaire

qui décrit les systèmes hétérogènes contenant à la fois les composants matériels et logiciels avec une bibliothèque de composants C++. Cette dernière fournit également des fonctionnalités d'association des composants logiciels aux composants matériels. Le format SERIF est par la suite interprété par un ensemble d'outils [Gue07], pour produire le code logiciel exécutable pour chaque processeur de la plateforme. Cet environnement est appelé APES (APplication Elements for System-on-chips) [Apes]. Il s'agit d'un environnement qui rassemble plusieurs composants logiciels (pilotes, systèmes d'exploitation, librairies de communication, librairies de gestion des tâches applicatives, etc.) que nous pouvons facilement rassembler et compiler pour générer rapidement le logiciel embarqué exécutable pour les architectures multiprocesseurs hétérogènes.

Gaspard2 (Graphical Array Specification for Parallel and Distributed Computing) [Bou03] [Gam10] est proposé par l'équipe DaRT de l'INRIA et par le laboratoire d'informatique fondamentale de Lille (LIFL). Il s'agit d'un environnement qui permet la génération de code logiciel et matériel en vue de déployer des applications de traitement du signal intensif sur des architectures multiprocesseurs, massivement parallèles et à mémoire partagée. Ce type de plateformes et d'applications est très différent de ce que nous ciblons dans cette thèse. Ce flot prend en entrée un modèle fonctionnel de l'application, un modèle d'architecture et un modèle d'allocation des ressources. Les modèles de haut niveau sont spécifiques à la description des applications de traitement de signal intensif et des architectures massivement parallèles. Les applications gèrent les données dans des tableaux multi dimensionnels, auxquels des tâches accèdent en entrée et en sortie. Le déploiement de ce type d'application sur les architectures massivement parallèles est traduit par l'association des tâches et leurs ordonnancements sur les unités de calcul, et des données sur des sections mémoires bien définies. Initialement, un profil UML pour Gaspard2 a été défini [Cuc05], pour décrire les caractéristiques du langage Array-OL [Dem95]. Dans les versions récentes de l'environnement Gapsard2, le standard MARTE [Marte] de l'OMG (Object Management Group) a été introduit pour la représentation de haut niveau des applications, des architectures et du partitionnement [Pie08] [Bou07]. De nombreux concepts, tels que la répétitivité structurelle, ont été ajoutés au profil MARTE pour supporter les caractéristiques des applications et des architectures traitées dans l'environnement Gaspsard2.

Le logiciel Matlab Simulink est utilisé dans [Han09] [Hua07] pour le développement du logiciel binaire à exécuter par des processeurs. Il offre un environnement adapté pour la modélisation des algorithmes applicatifs. Dans [Pop08] [Pop09], une approche de génération de code logiciel, basée sur la modélisation de haut niveau de l'application avec Simulink est proposée. Le code binaire est généré d'une manière graduelle à travers plusieurs niveaux d'abstraction pour faciliter son débogage. Dans [Han09], un modèle fonctionnel est produit en Simulink, dans une première étape, à partir d'une spécification initiale de l'application. Un modèle CAAM (Combined Architecture Algorithm Model), également décrit en Simulink, est produit dans une seconde étape. Ce modèle rassemble les éléments applicatifs et les composants de l'architecture cible. Il est ensuite transformé pour obtenir un modèle intermédiaire écrit en XML, appelé COLIF [Ces01]. Ce modèle est utilisé par un générateur de l'architecture matérielle pour produire des modèles de simulation des architectures matérielles à différents niveaux d'abstraction. Un dernier générateur exploite enfin ce modèle pour générer le code binaire à valider par les modèles de simulations générés.

3.2.2. Synthèse

A ce stade, nous avons listé les principaux flots de génération de code logiciel exécutable par les processeurs. Chacun cible une famille d'applications et d'architectures spécifiques et repose sur des primitives de communication fournie sous forme de code exécutable par un processeur. Les générateurs de code associés à ces flots ne sont pas adéquats pour produire le code de configuration que nous cherchons à développer. En effet, ces flots génèrent un code pour les processeurs dont la structure est illustrée dans la figure 2.6. Ce code généré ne peut pas être exécuté par les interfaces réseau configurables que nous ciblons.

D'un autre coté, la plupart de ces flots se basent sur un modèle de représentation des applications qui respecte le modèle de calcul KPN ou qui représente l'algorithme applicatif. Or, le modèle de processus de Khan ne permet pas la représentation de l'arbitrage des transferts de données que nous cherchons à modéliser. En plus, nous avons besoin de décrire uniquement les flux des données sans algorithme applicatif puisque nous traitons des architectures figées à base d'IPs.

Par ailleurs, le modèle de représentation du partitionnement des ressources dans les flots présentés est dépendant des applications et des architectures, comme dans Gaspard2, par exemple. Dans une architecture multiprocesseur, il est évident d'associer les tâches applicatives aux processeurs et les canaux de communications aux composants de mémorisation. Ce type de partitionnement ne s'adapte pas au type de plateforme que nous traitons. En effet, nous cherchons à représenter un partitionnement des ressources de calcul et de communication logicielle sur des composants matériels comme l'interface réseau configurable.

Pour toutes ces raisons, nous pensons que les flots de génération de code logiciel exécutable par les processeurs ne sont pas exploitables pour générer le code de configuration. En effet, ni les générateurs, ni les modèles de haut niveau ne répondent aux besoins des composants matériels tels que les interfaces réseau configurables.

3.3. Conclusion

Nous avons présenté, dans ce chapitre, les deux uniques travaux (Æthereal et Magali) traitant les interfaces réseau configurables. Pour générer le code de configuration, les flots proposés dans ces deux exemples, y compris les modèles de haut niveau, sont spécifiques aux plateformes cibles. Par ailleurs, il manque à ces solutions la formalisation et la structuration du code de configuration pour faciliter son développement.

En outre, nous avons montré, dans ce chapitre, que les flots de génération de code logiciel pour les processeurs ne sont pas adéquats à la génération du code de configuration.

Ainsi, nous proposons dans cette thèse une approche commune de génération du code de configuration qui ne dépend pas de la plateforme cible. Afin de faciliter son développement, nous organisons le code de configuration dans une structure en couches et nous automatisons sa génération.

Chapitre 4. Définition d'un flot de génération de code de configuration supportant les interfaces réseau configurables

4.1. Rappel de la problématique 41
4.1.1. Problèmes conceptuels 41
4.1.2. Problèmes liés aux approches de génération de code 43
4.2. Proposition d'une structure en couches du code de configuration 44
4.2.1. Présentation de la structure en couches du code de configuration 44
4.2.2. Comparaison de l'organisation en couches du code de configuration avec la pile logicielle d'un processeur 47
4.3. Définition d'un flot de génération de code de configuration 48
4.3.1. Besoins pour la définition d'un flot de génération de code de configuration 49
4.3.2. La partie avant du flot (Front-End) 49
4.3.3. La partie Arrière du flot (Back-End) 50
4.4. Modèle de représentation des plateformes à base d'IPs autour d'un réseau sur puce à interfaces réseau configurables 51
4.4.1. Abstraction des interfaces réseau configurables 52
4.4.2. Les chemins de communication physiques 53
4.5. Modèle de représentation des applications orientées flot de données 54
4.5.1. Présentation 54
4.5.2. Les canaux de communication multi entrées multi sorties 55
4.6. Technique de partitionnement des ressources 57
4.6.1. Présentation de la technique de partitionnement 57
4.6.2. Contrat de partitionnement 58
4.6.3. Modèle de partitionnement des ressources 59
4.7. Modèles intermédiaires et génération de code de configuration 59
4.7.1. Modèle intermédiaire général 60
4.7.2. Modèle intermédiaire spécifique 60
4.8. Conclusion 62

La première contribution de cette thèse est la proposition d'un flot de génération de code de configuration adapté aux NoCs ayant des interfaces réseau configurables. A travers cette contribution, nous proposons et exploitons des représentations de haut niveau des applications et des architectures matérielles pour générer une partie ou la totalité du code de configuration (fichiers de configuration, paramètres, microprogrammes, etc.). Les différentes étapes de ce flot seront détaillées dans ce chapitre.

4.1. Rappel de la problématique

Dans ce paragraphe, nous rappelons les problématiques que nous avons soulevées au chapitre 2 et auxquelles nous voulons apporter des réponses dans cette thèse. Le flot classique de génération de code logiciel, présenté en 2.2.2 et illustré par la figure 2.7, est spécifique aux processeurs. Nous avons vu que ce flot était insuffisant pour la génération de code de configuration pour les architectures que nous ciblons. Pour préciser les problèmes à traiter dans ce chapitre, nous les classons en deux catégories : les problèmes conceptuels et les problèmes relatifs aux approches de génération de code.

4.1.1. Problèmes conceptuels

Les problèmes conceptuels sont liés à la modélisation de haut niveau des applications, des plateformes et du partitionnement des ressources logicielles et matérielles. Ils concernent également l'organisation du code de configuration en une structure en couches, comparable à celle de la pile logicielle présentée en 2.2.1.

Pour décrire les applications orientées flots de données, les modèles utilisés dans les flots de génération de code logiciel classiques sont basés sur les modèles de calcul KPN et SDF principalement. Cela est insuffisant pour les applications que nous ciblons, où nous avons de nouveaux besoins comme la représentation des arbitrages dans les transferts de données entre les canaux de communications logiciels.

Pour générer le code de configuration, nous devons représenter l'aspect configurable des composants de calcul et de communication dans le modèle d'architecture. De même, nous avons besoin de décrire le chemin des données à travers le réseau. Pour cela, nous devons identifier les composants qui assurent la gestion des communications de bout en bout entre les IPs. Ainsi, l'intégration des chemins de communication dans le modèle d'architecture représente une nécessité. Il faut noter que ces chemins sont très peu décrits dans les langages existants.

Le déploiement des applications est réalisé à travers la configuration des interfaces réseau. Ces dernières fournissent des services comme le contrôle de flux et la synchronisation des communications et des exécutions des tâches dans les IPs. Afin de mener à bien la génération des configurations nécessaires, nous avons besoin de représenter les paramètres configurables des interfaces réseau. La valeur de ces paramètres dépend des informations applicatives de communication et de calcul. Ainsi, il est indispensable de porter les informations décrivant le trafic des données applicatives sur les interfaces réseau. Ce type de partitionnement n'est pas possible dans le flot classique de génération de code, qui permet uniquement l'association des tâches applicatives aux processeurs, et des canaux de communication logiciels aux composants de mémorisation.

Nous exprimons les besoins de génération de code de configuration dans la figure 4.1. Pour cela, nous opposons le flot de génération de code logiciel embarqué classique (Fig. 4.1.A), au flot de génération de code de configuration (Fig. 4.1.B) qui répondrait aux exigences des plateformes à base de NoCs configurables.

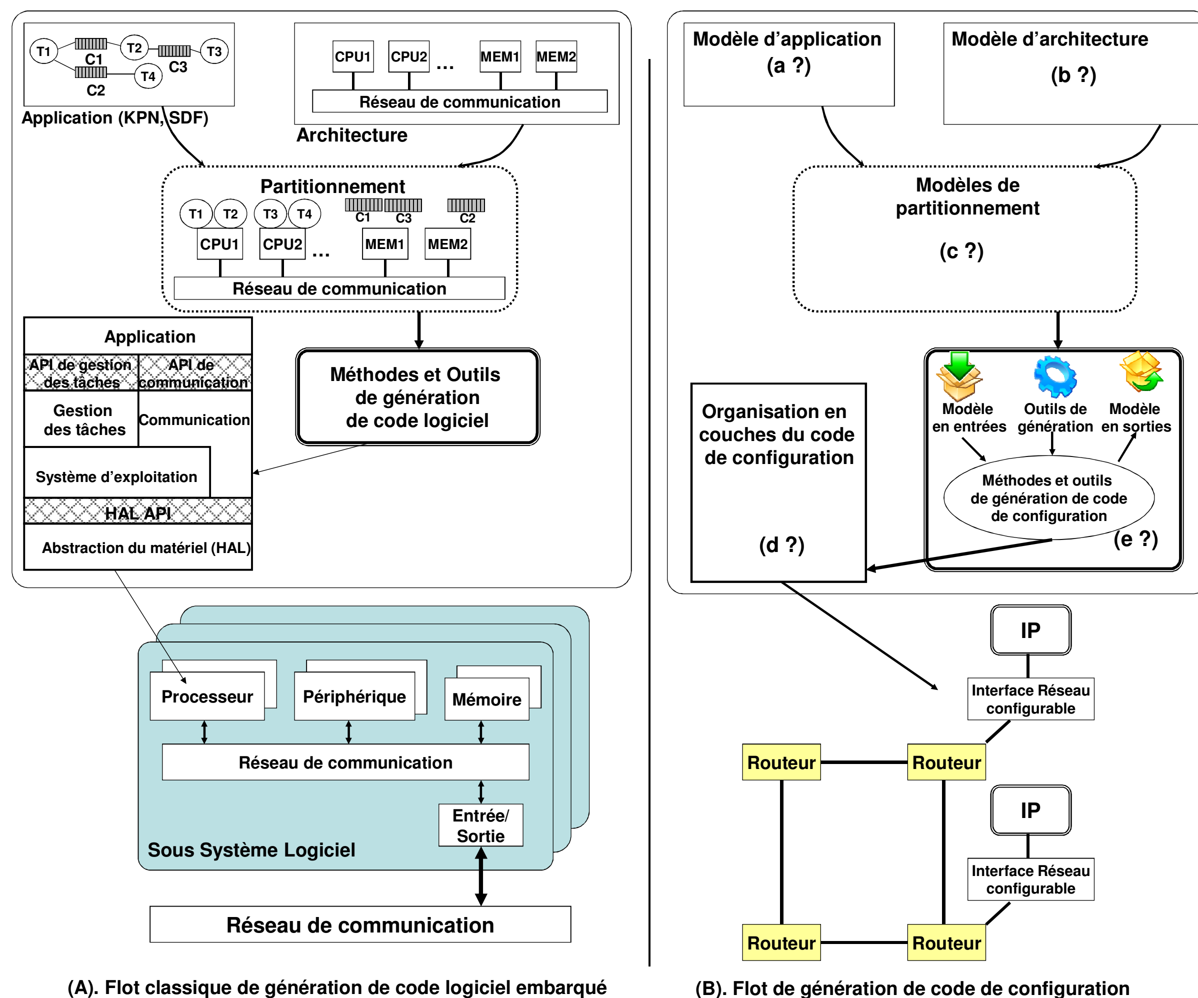


Figure 4.1. Rappel des problématiques à traiter

Cette figure résume les problèmes auxquels nous nous confrontons : Nous avons besoin de déployer des applications contenant beaucoup d'arbitrages de données entre les canaux de communications logiciels (Fig. 4.1.a) sur des plateformes. Ces dernières contiennent des interfaces réseau configurables (Fig. 4.1.b) offrant des services de synchronisation des tâches et de contrôle de flux. Elles doivent donc être configurées sur la base des caractéristiques des tâches applicatives et des échanges de données. Afin de réaliser ce déploiement, nous avons besoin d'une nouvelle représentation du partitionnement des ressources pouvant traiter ce nouveau genre d'applications et d'architectures (Fig. 4.1.c).

Pour faciliter la génération et le développement du code de configuration, il est classique d'opter pour son organisation en un ensemble de couches. La pile logicielle, détaillée en 2.2.1, en est un exemple. Cependant, cette organisation en couches, est spécifique aux sous systèmes logiciels, et n'est pas valable pour des architectures à base d'IPs, incluant

des interfaces réseau configurables. En effet, la pile logicielle modélise uniquement le logiciel à exécuter par les processeurs de la plateforme. Or dans les architectures visées, telles que Magali, il n'y a pas de processeurs qui jouent ce rôle. Ce sont les interfaces réseau qui sont chargées de synchroniser les tâches applicatives exécutées par les IPs. Nous devons donc formaliser le code de configuration des interfaces réseau et non le code logiciel exécuté par les processeurs dans une architecture classique. Nous adoptons pour cela une nouvelle organisation en couches (Fig 4.1.d) que nous détaillons dans la suite de ce chapitre.

4.1.2. Problèmes liés aux approches de génération de code

Une partie des problèmes qui se posent dans les approches de génération de code sont liés à la définition des différentes étapes à suivre ; depuis la modélisation de l'application, de l'architecture et du partitionnement des ressources jusqu'à la génération du code final.

Pour combler le fossé qui sépare les modèles de haut niveau du code final, l'utilisation de modèles intermédiaires représente une solution clé et répandue. L'utilisation d'un seul modèle intermédiaire implique d'inclure les informations spécifiques à la plateforme, tôt dans le flot, et rend ce modèle proche du code final. Or, nous cherchons à définir une approche générique capable de produire le code de configuration pour diverses plateformes. Le fossé reste alors entre les représentations de haut niveau d'un coté et le modèle intermédiaire, d'un autre coté. C'est pour cela qu'une solution à deux modèles intermédiaires peut être envisageable. Le premier est dit général et indépendant de la plateforme ciblée. Le deuxième est dit spécifique et il est obtenu par raffinement du premier. L'approche de génération de code de configuration (Fig.4.1.e) est ainsi définie en se basant sur ces deux modèles intermédiaires. Chaque phase du flot est centrée sur l'un d'eux.

En ingénierie dirigée des modèles (MDE : Model Driven Enginnering), les deux modèles intermédiaires que nous avons évoqués peuvent être considérés comme étant un PIM (Platform Independent Model) pour le modèle intermédiaire général et un PSM (Platform Specific Model) pour le modèle intermédiaire spécifique. Les modèles « PIM » et « PSM » sont des notions générales en MDE et ne représentent pas de concepts particuliers par défaut. C'est toujours le concepteur qui définit les éléments à décrire dans ces modèles.

Dans la suite de ce chapitre, nous détaillons le flot de génération de code de configuration que nous avons proposé. Nous avons eu recours à la représentation en modèles de haut niveau pour palier les problèmes liés :

- à l'inadaptation des modèles de calcul classiques pour la description des applications orientées flot de données supportant l'arbitrage des données,
- au besoin de description des interfaces réseau configurables et des chemins de communication, dans les modèles de description d'architectures,
- au problème de spécificité des modèles de partitionnement des ressources aux types d'application et aux plateformes ciblées.

Nous proposons d'abord une modélisation en couches du code de configuration qui permet de faciliter son développement.

4.2. Proposition d'une structure en couches du code de configuration

Pour structurer le code de configuration, nous proposons une organisation en trois couches principales. Elle peut s'étendre à quatre couches selon le mode de contrôle des différents IPs de la plateforme.

4.2.1. Présentation de la structure en couches du code de configuration

Dans les plateformes matérielles que nous étudions, l'exécution des applications est répartie entre les différents IPs. Un mécanisme de contrôle est responsable de la gestion et la synchronisation des communications et des tâches. Ce mécanisme est assuré par plusieurs composants matériels (processeur, interfaces réseau configurables, IPs, etc.), selon les fonctionnalités qu'ils fournissent.

Dans les plateformes où les IPs et les interfaces réseau ne fournissent pas de fonctionnalités de synchronisation et de séquencement des communications et des tâches, le contrôle de la plateforme se fait d'une manière centralisée à travers un processeur par exemple. Dans ce cas, le contrôle du système peut être modélisé par une seule couche nommée « Ordonnancement global » (Figure 4.2.a).

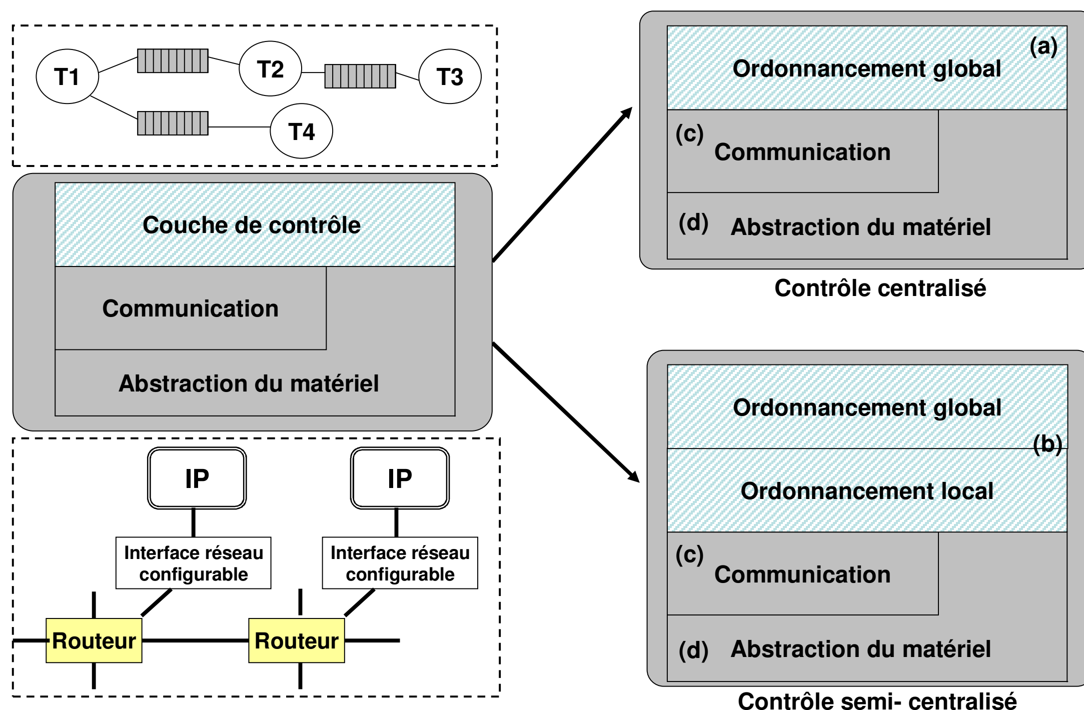


Figure 4.2. Organisation en couches du code de configuration

D'autre part, ces services de contrôle peuvent être fournis d'une manière hiérarchique, à travers un processeur central orchestrant des scénarii d'exécution locaux, relatifs aux IPs. Ce type de contrôle est appelé semi-centralisé. Il correspond à ce que propose la plateforme Magali à travers son processeur unique et ses interfaces réseau configurables.

Ce genre de contrôle peut être modélisé en deux couches nommées « Ordonnancement local » et « Ordonnancement global » (Figure.4.2.b). La première modélise le contrôle local des IPs effectué par les interfaces réseau configurables. Elle définit la synchronisation des communications et des tâches pour chaque bloc IP de la plateforme, y compris les IPs multi fonctions. La deuxième fournit les mécanismes de contrôle des scénarii d'ordonnancement locaux des IPs.

La structure en couches que nous proposons est composée de deux autres couches basses communes aux plateformes fournissant des mécanismes de contrôle centralisé ou semi-centralisé :

- La couche de communication (Figure.4.2.c) contient l'ensemble des configurations de communication. Celles-ci modélisent les transferts de données entre les IPs en précisant des détails à propos du chemin à suivre à travers le NoC, du nombre de données à transférer, etc. Les configurations s'exécutent généralement par le réseau de communication et en particulier par les interfaces réseau configurables.

- La couche d'abstraction du matériel (Figure.4.2.d) définit les fonctionnalités fournies par les blocs IPs. Elles sont décrites par un ensemble de paramètres qui configurent les registres des IPs. Cette couche intègre également les primitives (ou jeu d'instructions) de contrôle et de synchronisation des configurations de communication et des tâches. Ces dernières sont appelées dans des microprogrammes qui décrivent le scénario global ou les scenarii locaux d'exécution de l'application.

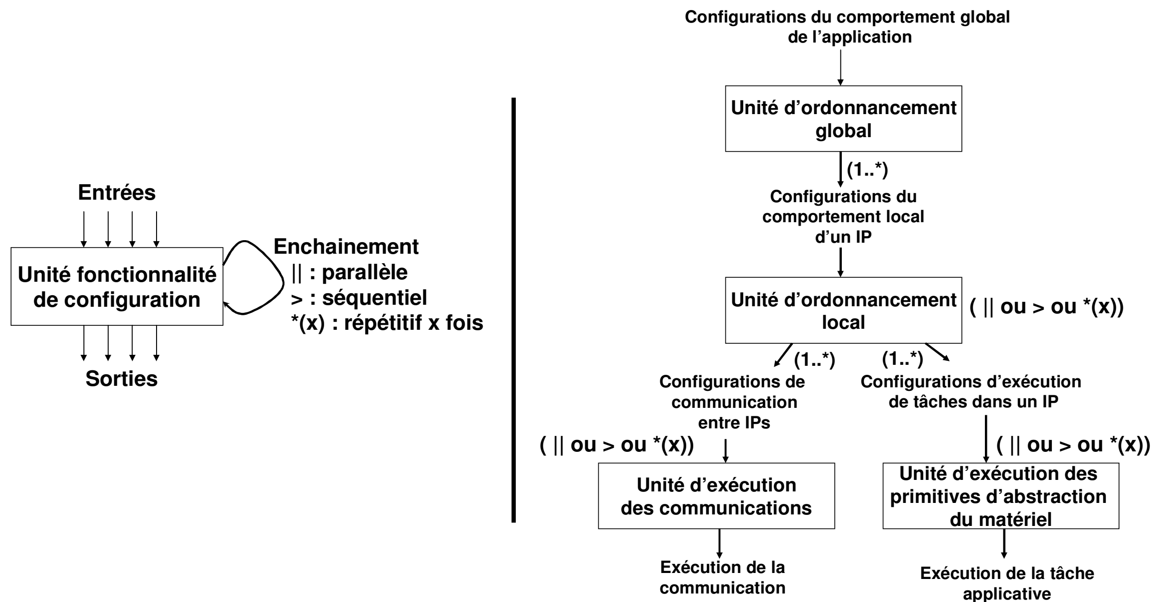


Figure 4.3. Modélisation des unités fonctionnelles de configuration et du code de configuration

Afin de donner un meilleur aperçu de l'organisation en couches du code de configuration, nous décrivons dans cette partie les types d'interactions entre ses différents niveaux. Nous définissons pour chaque niveau, une unité fonctionnelle de configuration. Nous avons ainsi identifié 4 unités chargées, en partant du haut, de l'ordonnancement global, de l'ordonnancement local, de la gestion des communications et de la gestion du matériel.

Chaque unité peut être considérée comme une procédure qui prend des arguments en entrée et rend des résultats en sortie. Les unités fonctionnelles de configuration peuvent être ordonnancées parallèlement ou séquentiellement. Elles peuvent également être répétitives.

La figure 4.3 schématise les interactions entre les différentes unités de configuration. L'unité d'ordonnancement global prend en entrée le comportement global de l'application. Elle produit en sortie un ensemble de configurations représentant les scénarii des comportements locaux des IPs. Les unités d'ordonnancement local reçoivent en entrée ces configurations et produisent en sortie deux types de configurations. Le premier concerne celles relatives aux communications. Celles-ci sont par la suite traitées par l'unité d'exécution des communications. Le second concerne les configurations d'exécution des tâches sur les composants IPs. Ces dernières sont traitées par l'unité d'exécution des primitives d'abstraction du matériel.

Exemple :

Nous modélisons, avec la structure en couches proposée, le code de configuration qui sert au déploiement d'applications sur la plateforme Magali. Le rôle des unités fonctionnelles est réalisé par l'unique processeur de l'architecture et par les interfaces réseau.

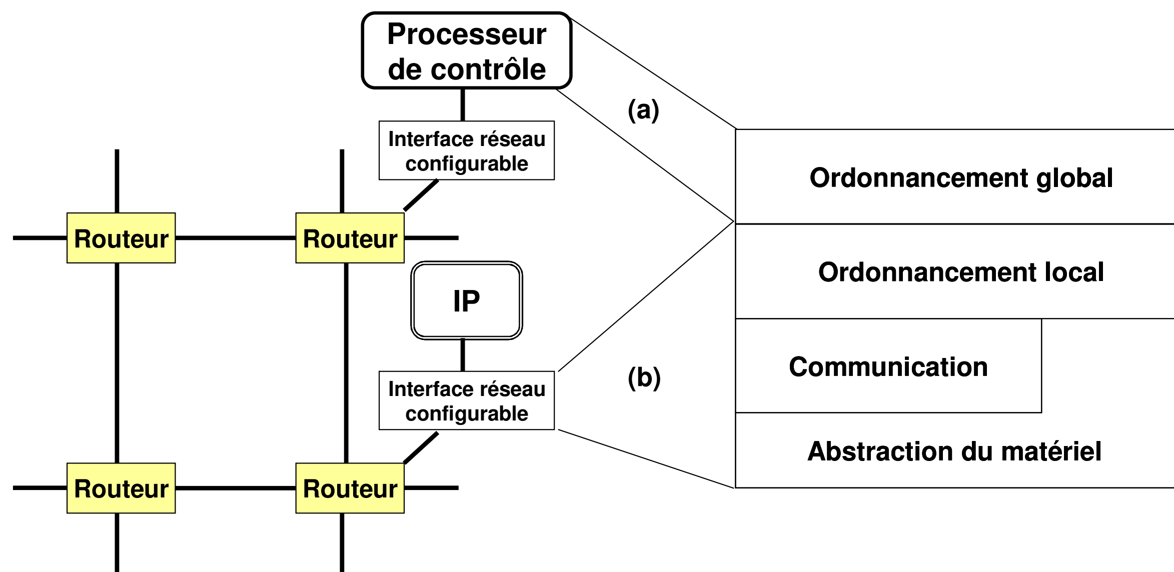


Figure 4.4. Exécution du code de configuration à contrôle semi-centalisé avec des interfaces réseau configurables

En effet, le processeur réalise l'ordonnancement global de l'application. Les interfaces réseau accomplissent les fonctionnalités des trois unités fonctionnelles des couches de niveau inférieur, qui sont l'ordonnancement local, l'exécution des communications et l'exécution des primitives du matériel (Figure.4.4).

Dans ce travail, nous adoptons l'organisation du code de configuration en 4 couches, afin de représenter le contrôle semi-centralisé du système. Nous considérons que la couche de

l'ordonnancement global est reliée à une ressource centrale qui peut être un CPU (comme dans le cas de la plateforme Magali Figure.4.4.a). Les fonctionnalités des autres niveaux sont exécutées par l'interface réseau qui contrôle les configurations élémentaires de communication et de gestion des IPs (Figure.4.4.b).

4.2.2. Comparaison de l'organisation en couches du code de configuration avec la pile logicielle d'un processeur

La figure 4.5 oppose l'organisation en couches du code de configuration (Fig. 4.5.B) que nous avons définie, à la structure de la pile logicielle des processeurs (Fig. 4.5.A). Dans cette section, nous dressons un comparatif entre les fonctionnalités des différentes couches des deux structures.

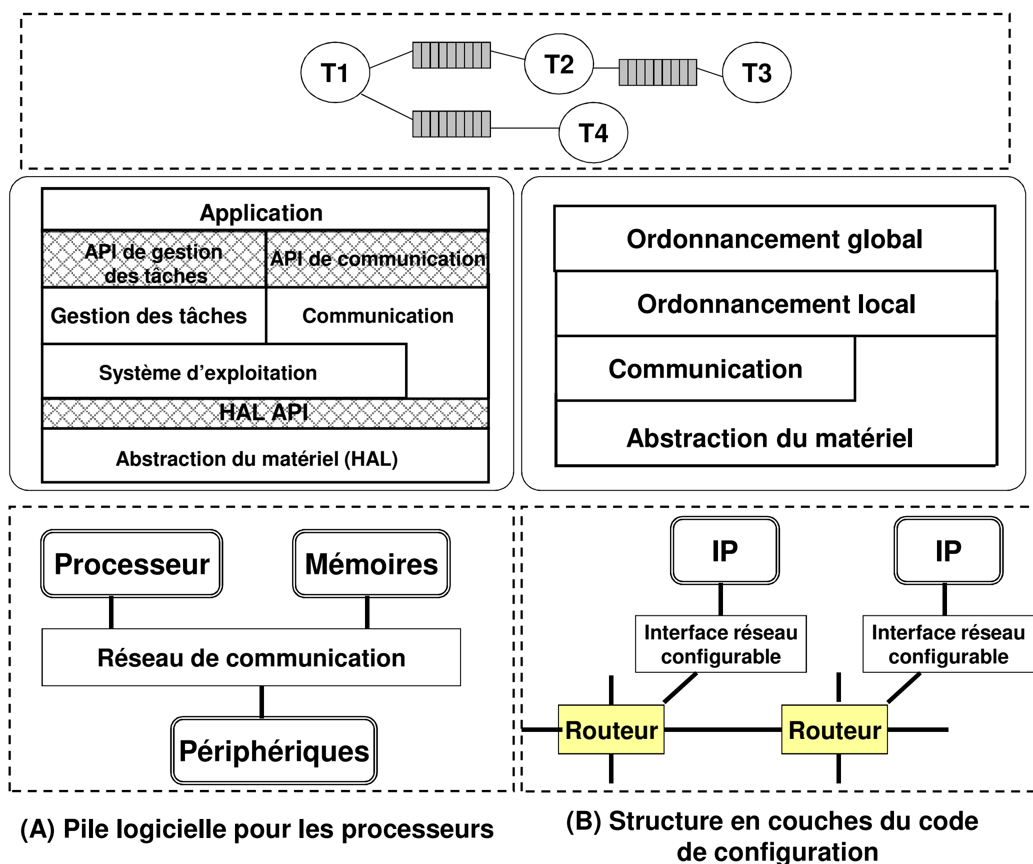


Figure 4.5. Comparaison entre la pile logicielle et la structure en couches du code de configuration

Dans le code de configuration, le comportement global de l'application est décrit dans la couche « Ordonnancement global ». Elle joue le même rôle que la couche « Application » dans la pile logicielle.

En effet, ces deux couches modélisent le comportement global de l'application. Pour la première, il s'agit de configurer le comportement global par orchestration des comportements locaux des IPs. Pour la seconde, il s'agit de programmer des tâches communicantes à travers des canaux de communication logiciels. Les couches « Ordonnancement global » et « Ordonnancement local » fournissent des mécanismes

d'ordonnancement des tâches et des communications pour les IPs. Cela correspond au rôle de l'ordonnanceur dans les systèmes d'exploitation. Celui-ci définit le séquencement des processus exécutés par le processeur.

La couche d'abstraction du matériel du code de configuration peut être assimilée aux couches d'abstraction du matériel de la pile logicielle (HAL API et HAL). Pour le code de configuration, on n'a pas besoin de séparer l'abstraction du matériel en une couche d'interface (API) et une couche de réalisation. En effet, le rôle de la couche HAL API est de permettre l'accès d'une manière transparente à des fonctionnalités implantées de différentes manières, dans la couche HAL, selon les caractéristiques du matériel. En revanche, dans les architectures que nous étudions dans cette thèse, les IPs fournissent des fonctionnalités matérielles spécifiques, c'est-à-dire qu'elles sont développées d'une manière unique sous forme de circuit électronique.

La couche de communication, dans les deux structures, fournit des primitives pour programmer les communications entre les composants de calcul. Dans la pile logicielle, elle permet de décrire les échanges entre des tâches applicatives associées aux processeurs. Elle est développée dans des bibliothèques logicielles. Dans la modélisation du code de configuration, elle englobe un ensemble de configurations qui spécifient les échanges entre les IPs. Ces dernières sont généralement associées au réseau sur puce, et en particulier aux interfaces réseau configurables.

La pile logicielle, contient une couche de gestion des tâches responsable de la synchronisation des tâches applicatives. Les fonctionnalités fournies par cette couche sont comparables à celles du niveau d'abstraction de matériel de la structure du code de configuration. En effet, dans les architectures à base d'IPs, les tâches sont distribuées sur les différents IPs. Leur gestion (création, suppression, etc.) se base sur des primitives très spécifiques et proches des composants matériels (jeu d'instructions), comme pour charger une configuration correspondante à une fonctionnalité d'un IP.

Pour la plateforme Magali, les primitives de gestion des tâches et des communications sont fournies par les interfaces réseau configurables. Ces primitives sont définies dans la couche d'abstraction du matériel et appelées par la couche de l'ordonnancement local pour décrire les scénarii de synchronisation locaux. Ces derniers sont ensuite exécutés par l'interface réseau.

Dans cette section, nous avons structuré le code de configuration en couches. Nous proposons dans la section suivante un flot de génération de ce type de code pour accélérer et automatiser son développement.

4.3. Définition d'un flot de génération de code de configuration

Nous avons mis en évidence au chapitre 2 l'inadéquation des flots de génération de code classiques, aux plateformes, telles que Magali, que nous étudions dans cette thèse. Nous proposons dans cette section un nouveau flot qui répond aux besoins de la génération de code de configuration.

4.3.1. Besoins pour la définition d'un flot de génération de code de configuration

Pour définir un flot de génération de code de configuration, nous avons besoin de certaines données en entrée : Un modèle décrivant les tâches applicatives et les flux de données échangées entre elles est nécessaire. Nous avons également besoin de savoir comment les tâches et les flux de données sont partitionnés par rapport aux IPs. Ainsi, un modèle de description de l'architecture matérielle et un modèle de partitionnement des ressources sont également nécessaires. Ce dernier doit être générique pour supporter plusieurs contraintes et scénarii de partitionnement liés aux caractéristiques des composants IPs et du réseau sur puce. En effet, le modèle de partitionnement doit supporter l'association des canaux de communication logiciels aux composants matériels qui fournissent les fonctionnalités de synchronisation des communications, à savoir les interfaces réseau.

Le passage de ces représentations vers le code de configuration nécessite plusieurs transformations. En premier, il faut rassembler les informations applicatives et d'architectures dans un modèle unique qui exprime, d'une manière structurelle, comment l'application est portée sur la plateforme. Ce modèle est enrichi, dans une seconde étape, avec des informations spécifiques au code de déploiement de l'application. Il s'agit d'exprimer, par exemple, la grammaire des configurations, des références vers des bibliothèques de composants logiciels (pilotes de périphériques, composants de simulation, adresses d'interruptions, etc.). Ces informations précises sont fournies par une tierce personne ayant une bonne connaissance de la plateforme cible et de la structure du code à produire. Elles servent à traduire le premier modèle en un second proche du code de configuration.

Nous spécifions ce processus de génération de code en deux phases distinctes. Nous proposons une approche « Front-End/Back-End », adoptée par les compilateurs. La première phase est indépendante de la plateforme cible alors que la deuxième lui est spécifique. La partie avant (« Front-End ») représente un point d'entrée commun pour programmer plusieurs types de plateformes configurables. La partie arrière («Back-End ») prend en considération les caractéristiques de la plateforme ciblée, à savoir la structure et la grammaire des configurations à générer. Elle contient également les outils de génération du code final. Chaque partie du flot est centrée autour d'un modèle intermédiaire qui favorise le passage graduel vers le code de configuration.

4.3.2. La partie avant du flot (Front-End)

La partie « Front-End », illustrée dans la figure 4.6.A, est le point d'entrée dans le flot de génération de code de configuration. Elle comporte la modélisation de l'application (Figure.4.6.a), de l'architecture (Figure.4.6.b), du partitionnement des ressources (Figure.4.6.c) ainsi que la génération d'un premier modèle intermédiaire.

Le partitionnement et le placement des ressources à ce niveau n'est pas automatique mais manuel. Il prend en considération différentes contraintes architecturales liées aux caractéristiques des composants IPs et des interfaces réseau configurables.

Des transformations sont appliquées, dans cette partie du flot, permettant de rassembler toutes les représentations de haut niveau et produire le premier modèle

intermédiaire appelé le modèle intermédiaire général (GIM pour « General Intermediate Model »). Il représente le déploiement de l'application à travers l'association des ressources applicatives aux composants matériels. Ce modèle intermédiaire est structurel. Il ne contient pas de détails sur la plateforme cible, et n'a pas de connaissance sur la structure du code à générer. Il est introduit pour assurer la portabilité de cette partie du flot afin de produire du code pour des plateformes cibles différentes.

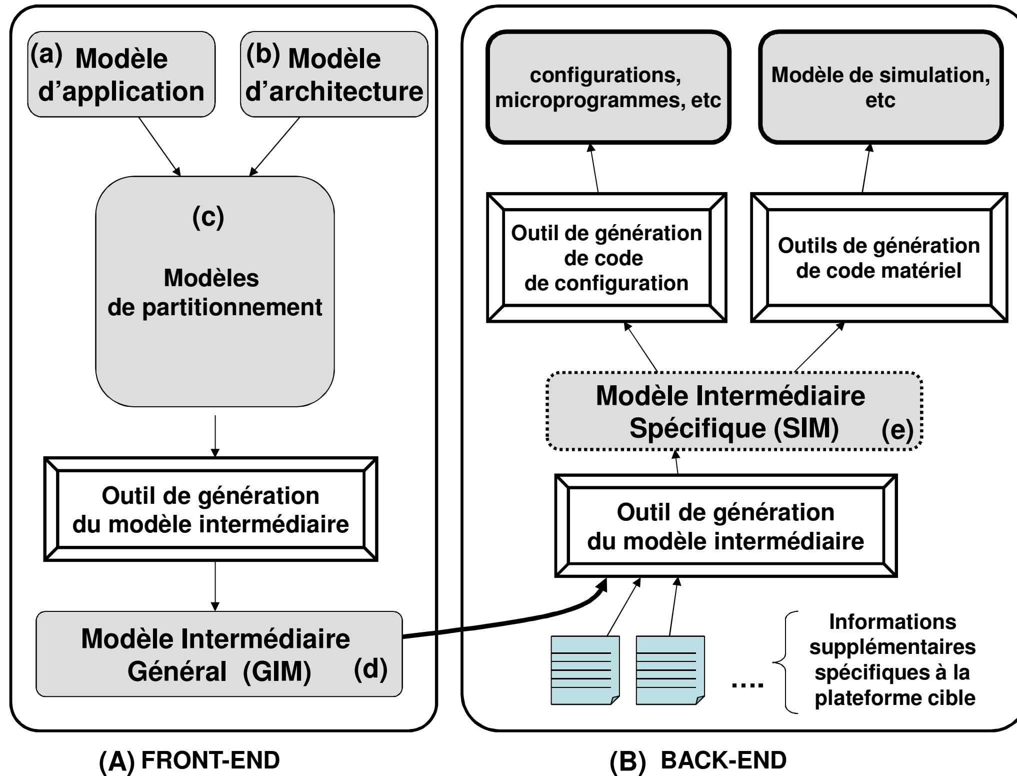


Figure 4.6. Flot da génération de code de configuration

4.3.3. La partie Arrière du flot (Back-End)

La partie « Back-End » (Figure.4.6.B) dans le flot de génération de code de configuration vient spécialiser la première phase pour une plateforme bien déterminée. En effet, la génération de code de configuration à partir du modèle intermédiaire général est quasi-impossible. Ce dernier ne fournit pas de détails concernant les paramètres à initialiser et le jeu d'instructions à utiliser, pour le contrôle des communications et des tâches. Il faut donc disposer d'un second modèle qui contient ces détails et complète le premier.

La seconde partie du flot prend en entrée le modèle intermédiaire général (GIM), en plus des informations spécifiques à la plateforme (la structure des fichiers de configuration, les primitives à utiliser dans les microprogrammes, etc.), fournies par les connaisseurs de la plateforme.

Des transformations sont appliquées pour produire le second modèle intermédiaire appelé modèle intermédiaire spécifique (Figure.4.6.e) (SIM pour « Specific Intermediate Model »). Il s'agit d'une représentation plus détaillée du déploiement de l'application et contenant des informations proches du code de configuration (paramètres et configurations

associées aux interfaces réseau, aux composants IPs, pilotes associés aux périphériques, etc.). Ce code final est obtenu suite à des transformations appliquées au modèle intermédiaire spécifique. Des modèles de simulation de l'architecture matérielle peuvent également être produits à partir du même modèle intermédiaire.

Nous allons, dans la suite, présenter les étapes de notre flot. Nous détaillons d'abord les modèles de représentation de l'architecture, de l'application et du partitionnement ; puis les modèles GIM et SIM.

4.4. Modèle de représentation des plateformes à base d'IPs autour d'un réseau sur puce à interfaces réseau configurables

Le flot de génération de code de configuration se veut générique dans sa partie avant. Il doit donc accepter un large éventail de plateformes. Pour cela, il faut lui fournir en entrée un modèle de représentation de l'architecture.

Ce modèle doit permettre de représenter l'ensemble des IPs et le support de communication configurable. Il doit rendre possible d'énumérer la liste des tâches qu'un IP peut fournir en précisant les paramètres de l'IP à initialiser. Ces détails nous aident dans la génération du code de configuration de calcul. Le modèle de la plateforme doit permettre également la description des caractéristiques de communication des IPs en offrant la possibilité au concepteur de représenter comment deux composants IPs échangent des données. Cela est possible en précisant les composants matériels qui assurent cette communication. Ces informations de communication nous aident dans la configuration des interfaces réseau qui se chargent d'acheminer les données.

Il ne s'agit pas de définir un nouveau langage. Ce modèle n'est qu'une représentation structurelle de la plateforme sans sémantique. Il existe plusieurs solutions pour représenter une plateforme matérielle, mais une solution qui émerge depuis quelques années est le standard IEEE 1685 (IP-XACT) [Ip-xact]. Ce standard a été défini pour décrire les composants IPs afin de faciliter leurs réutilisations et leurs intégrations dans des environnements de développement différents. Il est particulièrement bien adapté pour représenter la structure des plateformes. Malheureusement, pour notre flot de génération de code de configuration, nous avons besoin d'informations supplémentaires (chemins de communications, liste des fonctionnalités fournies par les IPs), ce qui n'est pas supporté d'une manière native dans la version la plus récente d'IP-XACT. Nous avons été amenés à ajouter des extensions à ce standard et à développer les outils associés. Ces extensions sont décrites dans le chapitre 5.

Avec cette représentation, nous pouvons décrire une architecture contenant des unités de calcul, des composants de communication, des composants de mémorisation et des périphériques (Figure 4.7).

Les unités de calcul représentables sont les processeurs et les blocs IPs. Le processeur, n'est pas utilisé pour exécuter des tâches mais pour assurer le contrôle global et la synchronisation des tâches distribuées sur les IPs. Les unités de calcul qui exécutent les tâches applicatives sont les blocs IPs. L'ensemble des tâches qu'un IP peut fournir et exécuter est décrit dans la couche HAL associée aux composants IP.

Les ressources de communication décrites par cette représentation sont les réseaux sur puce, en particulier les interfaces réseau et les routeurs.

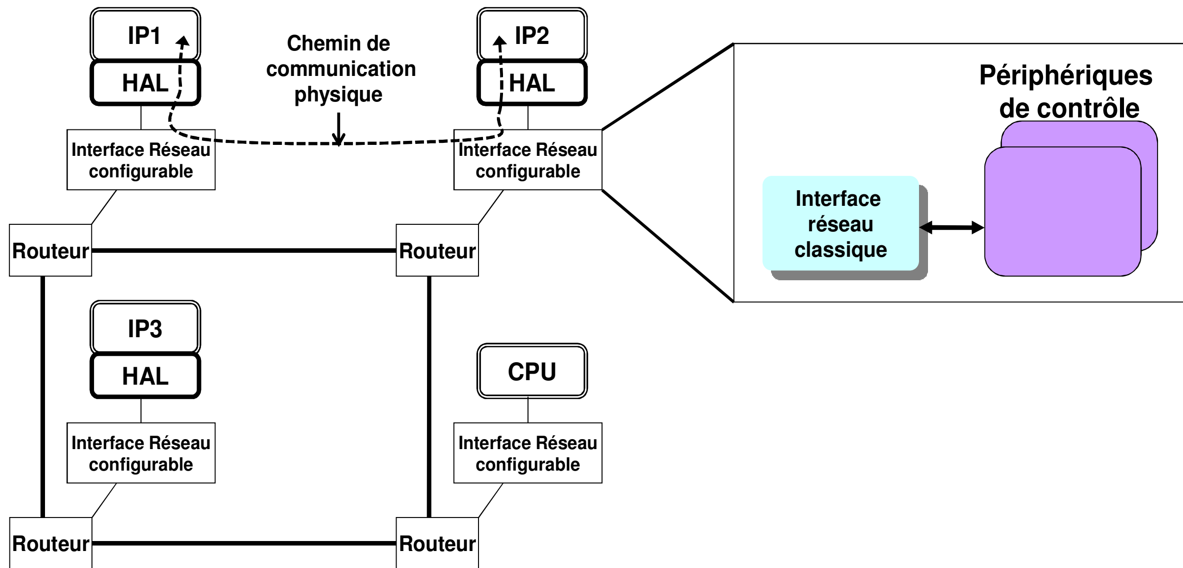


Figure 4.7. Représentation d'une plateforme configurable

Une interface réseau (NI pour Network Interface) configurable possèdent une interface, qui abstrait le protocole de communication entre les IPs et le réseau, et un comportement qui est programmable à travers des configurations. On peut alors le décomposer en une unité de contrôle et en un ensemble de périphériques pilotés par cette partie de contrôle. Les périphériques dans cette représentation étendent les possibilités de l'interface réseau classique. Ils apportent des fonctionnalités telles que la gestion des communications en entrée ou en sortie.

Dans les plateformes que nous traitons, les chemins de communication relient les IPs qui échangent les données à travers les routeurs et les interfaces réseau. Les périphériques de contrôle, inclus dans les interfaces réseau configurables, peuvent faire partie de la description des chemins de communication.

4.4.1. Abstraction des interfaces réseau configurables

La représentation d'une interface réseau configurable, qui fournit des fonctionnalités avancées de gestion de communication et de calcul, est définie par l'association d'une interface réseau et d'un ensemble de composants périphériques. Ces derniers permettent d'abstraire l'implantation matérielle des fonctionnalités avancées et du comportement configurable de l'interface réseau. Les périphériques peuvent, par exemple, effectuer la configuration et la synchronisation des communications et des tâches.

Exemple : Nous modélisons dans cet exemple les interfaces réseau configurables de la plateforme Magali. Nous rappelons que Magali inclut une unité nommée le gestionnaire de configuration et de communication (CCC). Cette dernière s'occupe de la gestion des communications entre les différents IPs ainsi que de la configuration des traitements et de leurs enchaînements.

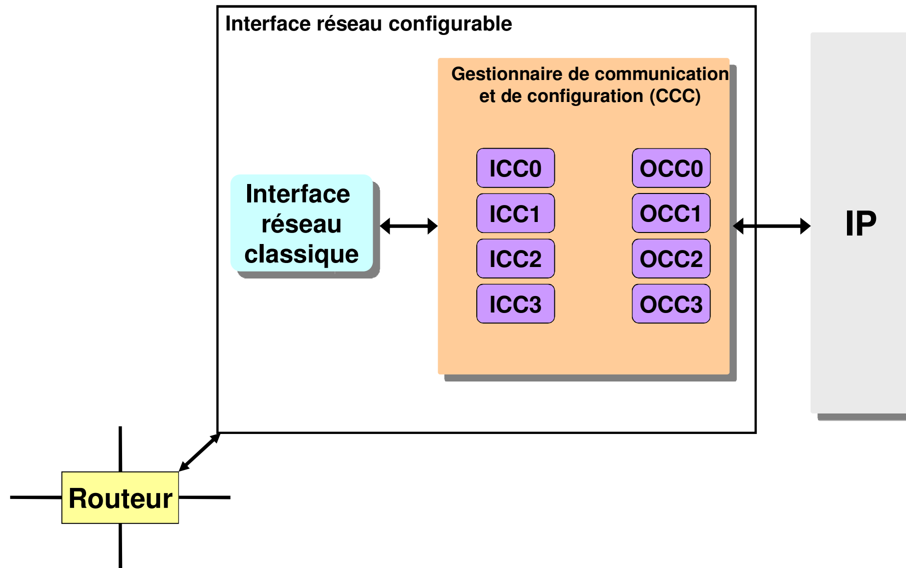


Figure 4.8. Modélisation de l'interface réseau configurable dans le réseau Magali

Le gestionnaire de communication et de configuration est modélisé, selon les cas d'utilisation, par au maximum, quatre contrôleurs de communication en entrée (ICC : Input Configuration Controller) et quatre contrôleurs de communication en sortie (OCC : Output Configuratin Controller). Chacun est considéré, dans la représentation que nous proposons, comme étant un périphérique de contrôle. Ce périphérique est interconnecté d'un coté à l'interface réseau et de l'autre au bloc IP (Figure4.8).

4.4.2. Les chemins de communication physiques

Un chemin de communication physique modélise les connexions entre des ressources matérielles qui échangent des données. Leur description sert, dans une autre étape du flot, à la génération du code de configuration de communication des interfaces réseau configurables.

Un chemin de communication est défini par un composant IP source, un destinataire, et une suite de ressources qui interviennent pour assurer cette communication. Ce sont généralement des composants de communication de type interface réseau ou routeur. Les périphériques associés aux interfaces réseau configurables peuvent faire partie de la liste des ressources qui réalisent les communications s'ils fournissent des services avancés de gestion de communication.

Exemple : Les contrôleurs de communication décrits comme étant des périphériques et interconnectés aux interfaces réseau, réalisent des fonctions d'acheminement des communications entre les IPs à travers le réseau. Ainsi un chemin de communication physique est décrit par deux IPs jouant le rôle des composants source et destinataire. La liste des composants qui assurent cette communication comporte un contrôleur de communication en sortie (OCC) associé à l'IP source, l'interface réseau liée à l'IP source, l'interface réseau liée à l'IP destinataire et un

contrôleur de communication en entrée (ICC) associé à l'IP destinataire :

composant IP source → OCC → NI → Suite de routeurs → NI →ICC → composant IP destinataire. (Fig.4.9)

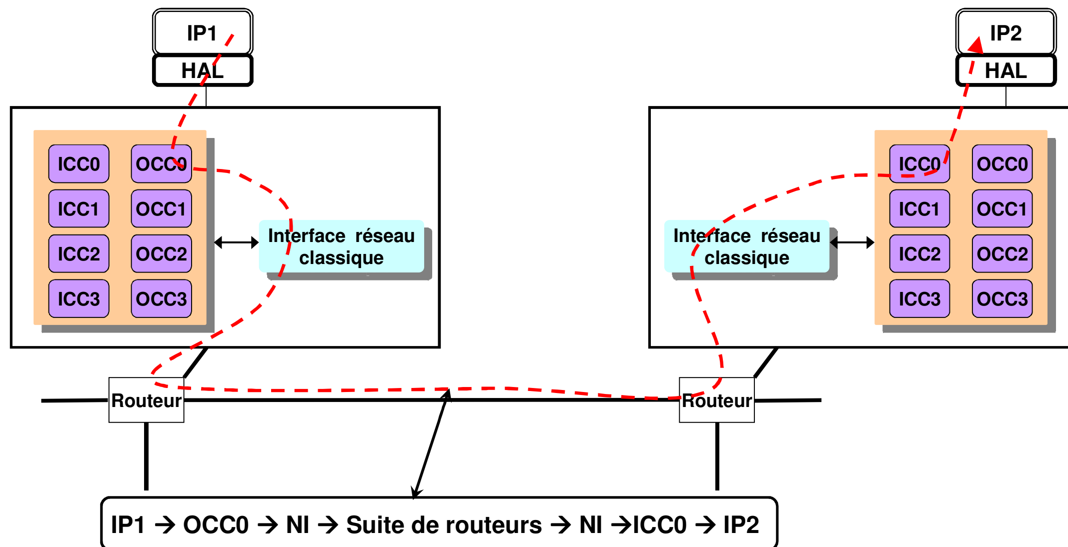


Figure 4.9. Représentation d'un chemin de communication physique dans la plateforme Magali

4.5. Modèle de représentation des applications orientées flot de données

Nous rappelons que le flot de génération de code de configuration se veut générique dans sa partie avant. Il doit donc accepter plusieurs types d'applications. Il faut donc lui fournir en entrée un modèle de représentation de l'application.

Ce modèle doit permettre de représenter des applications orientées flot de données dans lesquelles nous pouvons décrire des tâches parallèles communicantes avec des possibilités d'arbitrage de données. Il doit conserver la représentation des applications existantes en KPN ou SDF. Des informations concernant les quantités des données échangées entre les tâches applicatives doivent être décrites dans la perspective de configurer les interfaces réseau.

D'autre part, nous cherchons à représenter la quantité de données échangées, quand elle est variable, pendant l'exécution de l'application. Cet aspect dynamique de l'application se traduit, dans le code de configuration, par un paramètre variable dont la valeur est connue uniquement au moment de l'exécution.

4.5.1. Présentation

Pour répondre aux exigences de modélisation des applications, qui sont évoquées dans le paragraphe précédent (arbitrage des données, informations sur le trafic des données statique et dynamique, etc.), nous introduisons dans ce paragraphe, un modèle de représentation à haut niveau des applications orientées flot de donnée. Ce dernier intègre les modèles de calcul classiques (KPN, SDF). De plus, il permet de décrire les échanges de données entre les tâches

avec des possibilités d'arbitrage. Il ne s'agit pas d'un nouveau formalisme ou langage, mais d'une représentation textuelle en XML, décrivant l'enchainement du flot de données applicatif.

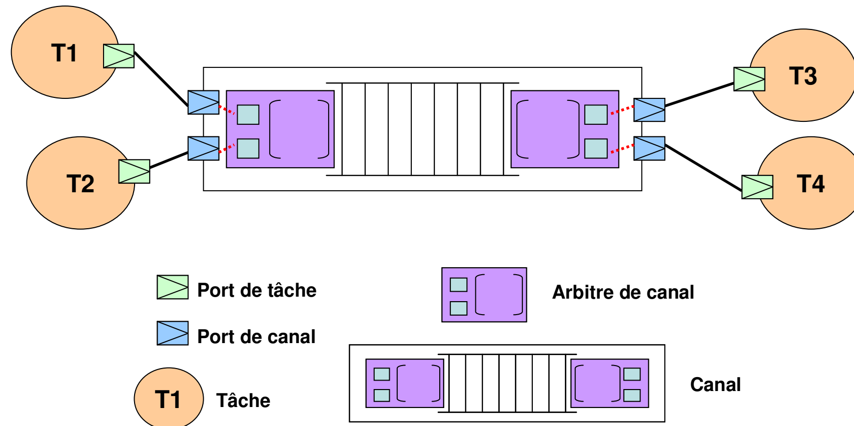


Figure 4.10. Composants de base pour la description des applications orientée flot de données

Une application décrite avec cette représentation, comme illustré en figure 4.10, est composée de deux types de composants : Les tâches et les canaux. Ces derniers représentent le support de communication entre les tâches. Un canal de communication, dans cette représentation, peut être multi lectures multi écritures. Il peut être de deux types : un canal FIFO ou un canal à base de mémoire partagée.

Une tâche accède aux canaux de communication via des ports de tâche. Un port de tâche correspond à l'appel, au sein du code de la tâche logicielle, à des primitives de communication ou à des primitives de gestion de la mémoire, représentée par le canal. Ce dernier peut ordonnancer ses accès en entrée ou en sortie via un arbitre.

Les tâches représentent les éléments qui contiennent le comportement de l'application. Le modèle décrit une tâche comme un nœud pouvant avoir plusieurs ports d'écriture et de lecture. Chacun peut être interconnecté à un et un seul port de canal. Un port de tâche est décrit par un nom, une direction (entrée ou sortie), la taille des données qui peuvent être transférées par ce port ainsi que le nombre d'occurrences de ce transfert. Cette dernière information peut traduire une boucle dans le code applicatif, dans laquelle on exécute une primitive d'écriture. Le port représentant cette communication a une valeur d'occurrences de transferts équivalente à la taille de la boucle.

La nouveauté que nous avons apportée dans cette représentation se situe au niveau de la description des flux de données par des canaux multi entrées multi sorties avec des possibilités d'arbitrage des données en entrée et en sortie. Nous détaillons ces canaux de communication dans le paragraphe suivant.

4.5.2. Les canaux de communication multi entrées multi sorties

Un canal de communication (Figure.4.11) peut disposer de plusieurs ports de lecture et d'écriture. Chaque port de canal est interconnecté à un port de tâche. L'accès au canal (en entrée ou en sortie) est défini avec un arbitre qui assure l'ordonnancement des accès aux ports du canal, grâce à la définition de matrices d'entrée et de sortie.

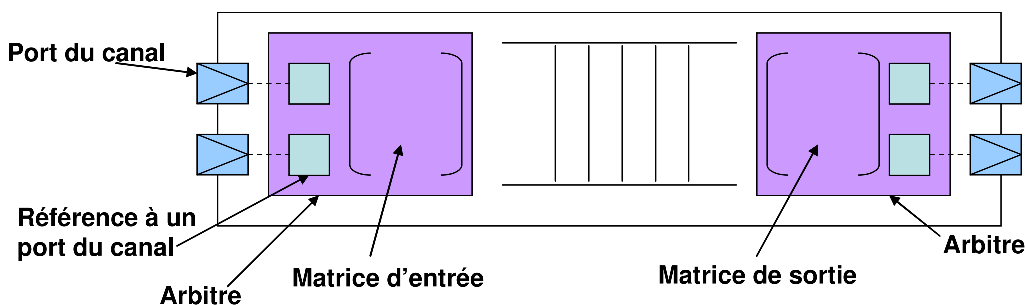


Figure 4.11. Composants de base d'un canal de communication multi entrée multi sortie

Une matrice d'entrée ou de sortie est définie pour chaque arbitre du canal. Elle décrit un ensemble de scénarii de lecture ou d'écriture de données en faisant référence aux ports du canal. Un scénario d'exécution est écrit dans une colonne de la matrice. La dernière valeur de la colonne correspond au nombre d'exécution de ce scénario. Le nombre de répétitions des scénarii, peut être fixe comme il peut être variable. Il prend plusieurs valeurs au cours de l'exécution de l'application, pour les flots de données dynamiques.

Exemple : La figure 4.12 illustre un exemple d'utilisation d'une matrice d'entrée composée d'un seul scénario d'exécution. La colonne est formée par 4 éléments. Les 3 premiers éléments correspondent au nombre d'écritures que les ports du canal vont effectuer. Le premier port du canal dont la taille de données est 10 va effectuer 4 écritures, puis le deuxième port va effectuer 2 écritures et pour finir, le troisième port effectue 3 écritures. Le dernier élément de l'unique colonne de la matrice d'entrée correspond au nombre d'exécutions du scénario précédemment décrit, à savoir une seule fois pour notre exemple. Pour des flots de données dynamiques, cette valeur peut prendre le nom d'une variable (variable « x » dans la figure 4.12) qui peut être initialisée ou calculée à plusieurs reprises durant l'exécution de l'application.

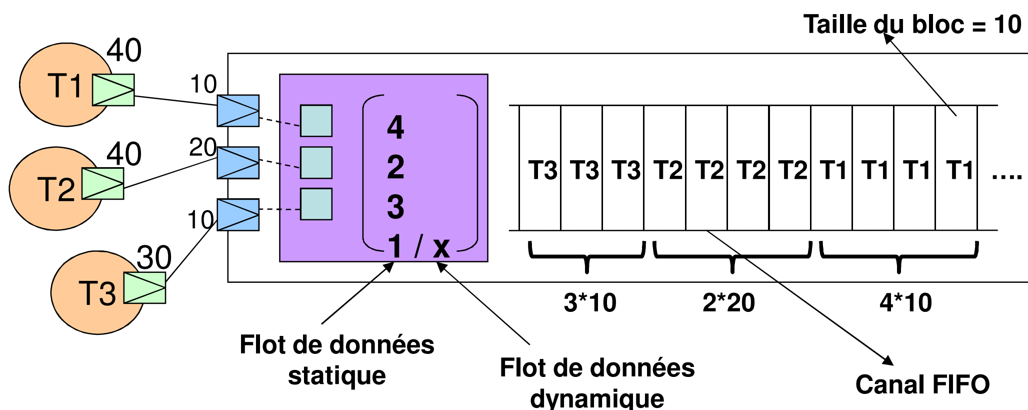


Figure 4.12. Exemple d'une matrice d'entrée dans un arbitre d'entrée

Nous pouvons voir sur le canal FIFO la disposition des données une fois la matrice d'entrée réalisée d'une manière statique (exécution du seul scénario une seule fois). La FIFO

```
est modélisée par des blocs de données de taille 10. Nous
voyons que les données provenant de la tâche T1 sont écrites
en premier en tête de la FIFO, puis les données provenant de
T2 et finalement les données provenant de T3. Les éléments de
sortie du canal (non modélisés dans la figure 4.12) qui sont
les ports du canal, transfèrent ces données vers d'autres
tâches de l'application.
```

La modélisation de l'ordonnancement des transferts de données dans le modèle applicatif est traduite par des configurations qui orchestrent les échanges entre les composants IPs de la plateforme. Pour générer ces configurations, les informations de trafic de données applicatives doivent être associées aux composants matériels adéquats dans les plateformes que nous ciblons. Le prochain paragraphe détaille les modèles de partitionnement ainsi que les modèles intermédiaires utilisés dans notre flot.

4.6. Technique de partitionnement des ressources

Le modèle de partitionnement de l'application sur la plateforme, que nous utilisons dans la partie « Front-End », doit respecter des contraintes liées aux caractéristiques de l'application et de l'architecture cible. Nous pouvons citer comme contrainte, l'interdiction du placement des tâches applicatives sur le processeur central de la plateforme Magali. Nous cherchons à exprimer également l'association des échanges de données entre les tâches aux interfaces réseau configurables. Pour garantir et exprimer ces contraintes, nous proposons un modèle que nous appelons le contrat de partitionnement. Nous garantissons, à travers des outils spécifiques fournis par le flot, que le placement des ressources respecte bien ces contraintes. Cela définit notre technique de partitionnement.

Nous rappelons que nous réalisons un partitionnement manuel. Nous ne faisons pas de calcul optimal et automatique de placement des ressources. Nous définissons, en utilisant cette technique, un modèle de partitionnement cohérent et exploitable pour la génération de code, pour des plateformes comme Magali.

4.6.1. Présentation de la technique de partitionnement

Nous avons évoqué dans les problématiques que les modèles de description du partitionnement sont souvent spécifiques aux plateformes cibles et aux types d'applications. Nous rappelons que les modèles de partitionnement des ressources sont souvent associés aux plateformes multi processeurs comme ceux de DOL [Thi07], DSX [Dsx], SDF3 [Stu06]. Or, pour les architectures à base d'IPs, le partitionnement ne se limite pas à un simple placement des tâches applicatives sur des composants IPs et des canaux de communications logiciels sur des espaces mémoires. En effet, les caractéristiques des composants matériels rentrent en compte lors de l'établissement des règles de partitionnement.

Dans ce type de plateformes, si les IPs échangent des données à travers des composants de mémoire partagée, les canaux de communication applicatifs peuvent leur être associés. Par ailleurs, si l'IP contrôle des communications, les informations d'ordonnancement du flot de données applicatif lui sont associées.

Si la gestion des communications est une fonctionnalité fournie par l'interface réseau configurable, comme dans Magali, le placement des informations de trafic de données est établi sur l'interface réseau.

Nous cherchons à définir un modèle de partitionnement qui palie le problème de dépendance des modèles classiques avec les caractéristiques de la plateforme. Nous avons alors introduit un nouveau modèle, que nous avons appelé méta partitionnement, qui se place en amont du partitionnement des ressources à proprement parlé (Figure.4.13).

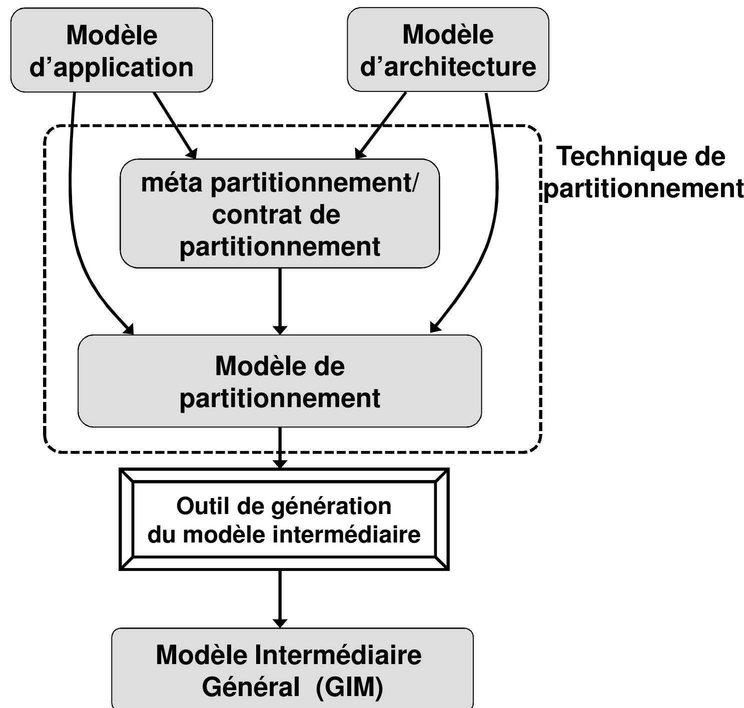


Figure 4.13. Méthodologie de partitionnement

Le méta partitionnement ou le contrat de partitionnement est une étape dans laquelle nous exprimons la manière de déployer une application sur une architecture. Il s'agit de décrire les règles de partitionnement que nous allons appliquer durant la phase de placement des ressources.

La diversité et la complexité des composants matériels (les IPs, les interfaces réseau, les périphériques de configuration et de contrôle, etc.) sont à l'origine de la diversité des modes de déploiement des applications. Le contrat de partitionnement est une nouveauté qui offre la possibilité de décrire le mode de déploiement en terme de mise en correspondance entre les types de composants applicatifs et les types de composants de l'architecture. Il s'agit d'un moyen de décrire des caractéristiques et des contraintes qui ne sont pas représentées ni dans le modèle d'architecture, ni dans le modèle d'application.

4.6.2. Contrat de partitionnement

Nous définissons le contrat de partitionnement par un ensemble de contraintes de deux types : les contraintes générales et les contraintes spécifiques.

Les contraintes générales spécifient les règles de placement des types de ressources applicatives (tâches applicatives, arbitres dans les canaux de communication) sur des types de ressources d'architecture (composants IP, interfaces réseau, périphériques de contrôle qui étendent les fonctionnalités de l'interface réseau).

> **Exemple** : Prenons comme architecture, une dont les interfaces réseau sont modélisées par des périphériques de contrôle qui sont les contrôleurs de communication « ICC » et « OCC », comme celles représentées en figure 4.8. Nous voulons indiquer qu'un arbitre de canal de communication de l'application ne puisse être mis en oeuvre en dehors d'un ICC ou d'un OCC. Le contrat de partitionnement devra donc exprimer cette règle qui devra être vérifiée lors du partitionnement.

Les contraintes spécifiques sont liées aux instances des composants matériels de l'architecture. Nous pouvons, par exemple, exprimer qu'un processeur présent dans une architecture à base d'IPs interconnectés par un réseau sur puce ne peut pas être en mesure d'exécuter des tâches applicatives. En d'autres termes, on n'a pas le droit d'associer des tâches applicatives à ce processeur qui est censé exécuter uniquement des fonctions de contrôle et d'ordonnancement.

On peut également exprimer une contrainte spécifique pour un composant IP, en listant l'ensemble des tâches qu'il peut exécuter. On interdit ainsi, l'association d'une autre tâche à cet IP lors de la phase de partitionnement.

4.6.3. Modèle de partitionnement des ressources

La définition d'un modèle de partitionnement nécessite trois entrées : le modèle de l'application, le modèle de l'architecture et le contrat de partitionnement. La mise en œuvre de ce modèle produit le premier modèle intermédiaire « GIM » qui englobe la description de l'architecture sur laquelle les éléments d'application sont déployés.

Le placement des tâches se fait par association de ces ressources applicatives sur les composants IPs interconnectés au réseau sur puce. De même, les canaux de communication applicatifs sont associés aux chemins de communication physiques. Ce placement doit préciser l'association entre un élément du canal applicatif (port des canaux, arbitres, la FIFO ou la mémoire partagée) et les ressources déclarées dans le chemin de communication (interfaces réseau, périphériques de contrôle, etc.).

4.7. Modèles intermédiaires et génération de code de configuration

Le flot de génération de code de configuration utilise deux modèles intermédiaires intitulés respectivement « GIM » et « SIM ». Le premier est indépendant de la plateforme. Il garantit la portabilité de notre solution. En revanche, le modèle « SIM » contient des informations spécifiques à la plateforme comme la grammaire et le vocabulaire du code de configuration.

4.7.1. Modèle intermédiaire général

Le modèle intermédiaire général (GIM) permet la description d'une application orientée flot de données déployée sur la plateforme cible, d'une manière structurelle. Il ne fournit pas de détails sur le vocabulaire spécifique et les primitives du code de configuration à générer.

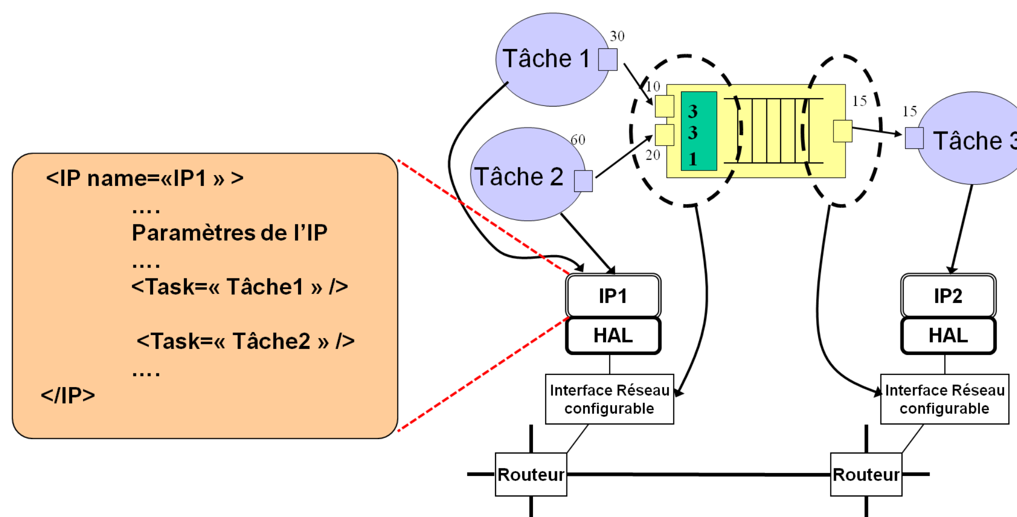


Figure 4.14. Modèle intermédiaire général

Le GIM décrit les composants d'architecture (IPs, interfaces réseau, périphériques de contrôle) auxquels sont associées les ressources applicatives (tâches, canaux de communication logiciels, arbitres, ports d'entrée et de sortie des canaux de communication, etc.) (Figure.4.14).

Ce modèle associe également les tâches applicatives aux IPs matériels. Il affecte les informations concernant le trafic des données applicatives (les arbitres à l'entrée et à la sortie des canaux, les ports des canaux de communications, les ports des tâches applicatives) aux interfaces réseau configurables et aux périphériques de contrôle (ICC et OCC).

D'autre part, les canaux de communication applicatifs sont mis en correspondance avec les chemins de communication physiques.

Le modèle GIM regroupe les informations applicatives et architecturales dans une représentation structurelle unique. Cette dernière est exploitable dans une prochaine étape pour la génération du code de configuration. Le fait de rassembler les informations applicatives et d'architectures dans un modèle unique facilite le développement d'outils et l'automatisation du flot de génération de code.

La structure XML de cette représentation est discutée dans le chapitre 5.

4.7.2. Modèle intermédiaire spécifique

Le modèle intermédiaire spécifique est obtenu suite à l'enrichissement du modèle intermédiaire général. Ce raffinement apporte de nouvelles informations sur le vocabulaire

des paramètres utilisés dans code de configuration ainsi et sur la structure du modèle de simulation.

Les informations spécifiques à la plateforme sont fournies par une tierce personne ayant une connaissance approfondie de l'architecture cible. L'interprétation du modèle GIM accompagné de ces informations donne comme résultat le modèle intermédiaire spécifique. Ce dernier est également un modèle structurel. Les informations spécifiques à la plateforme détaillent par exemple :

- Les bibliothèques de composants à utiliser pour la génération du modèle de simulation de l'architecture (SystemC, TLM, SoCLib, etc.).
- La structure des configurations à générer. Une configuration de communication, par exemple, doit être écrite en initialisant plusieurs paramètres : identificateur de la configuration, taille des données à transférer, chemin des données à travers le NoC, etc. Ce chemin doit être écrit en utilisant les mots clé « NORTH, SOUTH, EAST, WEST » indique le nombre de routeurs utilisés pour chaque transfert.
- Le jeu d'instructions (ou les primitives) utilisé pour le contrôle des configurations. Cette information sert à la génération des microprogrammes exécutés par l'interface réseau configurable.

Grâce à ces informations spécifiques, le modèle intermédiaire général est transformé en un modèle SIM proche du code de configuration à générer. Les informations applicatives de trafic de données (les arbitres, les ports de tâches, les ports de canal, etc.) associées aux périphériques de contrôle dans le modèle GIM, prennent une nouvelle forme dans le modèle SIM. Elles sont transformées en configurations qui respectent la syntaxe du code de configuration introduite par les informations spécifiques injectées au début de la phase arrière. La figure 4.15 donne un exemple de la structure d'une configuration de l'élément OCC de l'interface réseau. Elle respecte un vocabulaire spécifique à la plateforme Magali.

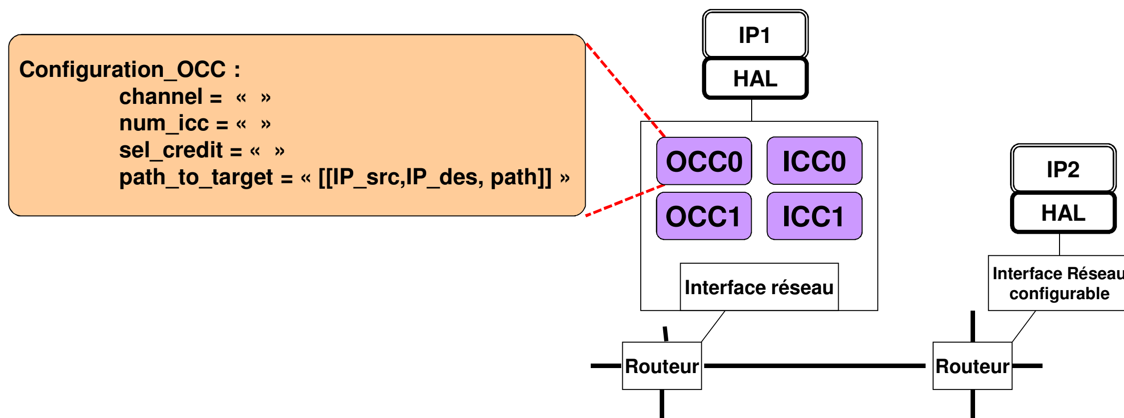


Figure 4.15. Modèle intermédiaire spécifique

Les paramètres décrivant les chemins des données sur le NoC sont déduits à partir des chemins de communications physiques du GIM, en plus de la syntaxe de ces chemins introduite dans les informations spécifiques. Des configurations de calcul sont également générées pour les composants IPs. Elles permettent de présenter les paramètres et les fonctionnalités de la couche HAL selon la syntaxe de la plateforme cible. D'autres

informations peuvent apparaître dans le modèle SIM se rapportant à la définition du plan mémoire des composants matériels, leurs adresses d'interruptions, la chaine d'outils à utiliser, etc.

Le modèle SIM est proche du code de configuration. Il spécialise le modèle GIM de la partie avant du flot et confirme la portabilité de cette dernière pour différents déploiements. Son utilisation facilite la génération de code de configuration et son processus d'automatisation.

La structure XML de la représentation SIM est présentée dans le chapitre 5.

Exemple : Prenons l'exemple d'un modèle GIM ayant un contrôleur de communication en sortie (OCC dans une interface réseau configurable) auquel nous avons associé un port d'une tâche applicative. Cette association est transformée en une configuration de communication relative au contrôleur (OCC), dans le modèle SIM (Figure 4.15). Elle décrit, avec la syntaxe du code de configuration cible, le nombre de données à transférer, le chemin de transfert à travers le réseau, l'identifiant du contrôleur de communication (ICC) qui va recevoir les données envoyées, etc.

Nous générons le code de configuration directement à partir du modèle SIM.

4.8. Conclusion

Dans ce chapitre, nous avons introduit une structure en couches du code de configuration, pour les plateformes que nous ciblons dans cette thèse. Nous avons défini les fonctionnalités fournies par chaque couche et nous avons établi une comparaison de cette structure avec la pile logicielle, spécifique aux architectures multiprocesseurs.

Nous avons introduit par la suite un flot de génération de code de configuration basé sur des représentations de haut niveau pour la description des applications, des architectures et du partitionnement des ressources. La première phase du flot est indépendante des détails de la plateforme cible alors que la deuxième lui est liée. Le flot est également basé sur deux modèles intermédiaires pour garantir la portabilité de notre approche de génération de code.

Le flot proposé peut être vu comme une application d'une approche MDE. La définition des représentations de haut niveau de l'application, de l'architecture et du partitionnement peut être considérée comme une définition des méta-modèles. Les modèles intermédiaires « GIM » et « SIM » peuvent être considérés respectivement comme des modèles « PIM » et « SIM ». Enfin, les outils de génération des modèles intermédiaires et du code final peuvent être considérés comme des outils de transformation de modèles.

Nous présentons dans le prochain chapitre la manière dont nous avons développé ce flot dans un environnement IP-XACT afin de bénéficier des mécanismes de réutilisation des composants et d'automatisation du flot.

Chapitre 5. Spécification du flot de génération de code de configuration dans un environnement IEEE 1685 (IP-XACT)

5.1. Rappel de la problématique ... 64
5.2. Présentation générale du standard IEEE 1685 (IP-XACT) ... 64
5.2.1. Les composants de base du standard IEEE 1685 ... 65
5.2.2. Intégration et enchaînement des outils dans IP-XACT ... 67
5.2.3. Outils de développement IP-XACT ... 68
5.3. Spécification du flot de génération de code de configuration basé sur le flot IP-XACT ... 68
5.3.1. Choix du standard IP-XACT ... 69
5.3.2. IP-XACT pour le flot de génération de code de configuration ... 69
5.4. Extension du standard IP-XACT pour le modèle de représentation ARDL ... 71
5.4.1. Extension de l'élément « component » pour la description des fonctionnalités des composants matériels ... 71
5.4.2. Extension de l'élément « design » pour la description des chemins de communication physiques ... 72
5.4.3. Extension de l'élément « designConfiguration » pour la représentation des informations spécifiques à la plateforme ... 73
5.5. Spécification du modèle de représentation de l'application APDL et des modèles de partitionnement ... 75
5.5.1. Modèle de représentation des applications orientées flot de données : APDL ... 75
5.5.2. Modèles de partitionnement ... 76
5.6. Automatisation du flot de génération de code de configuration ... 77
5.6.1. Les composants « generator » et « generatorChain » pour l'automatisation du flot de génération de code de configuration ... 78
5.6.2. Les modèles intermédiaires « GIM » et « SIM » ... 79
5.6.3. Synthèse ... 81
5.7. Intégration du flot de génération de code de configuration dans l'environnement Magillem ... 81
5.7.1. Présentation de l'outil Magillem ... 81
5.7.2. Construction d'une bibliothèque de composants matériels et assemblage des architectures ... 82
5.7.3. Environnement de développement de la partie « Front-End » du flot de génération de code de configuration ... 83
5.7.4. Développement des générateurs dans l'environnement Magillem ... 83
5.8. Conclusion ... 84

La deuxième contribution de cette thèse est la spécification du flot de génération de code de configuration dans un environnement IP-XACT (IEEE-1685). A travers cette contribution, nous mettons en œuvre le flot, et nous automatisons ses différentes phases. Ensuite, nous l'intégrons dans un environnement industriel nommé Magillem. Les différentes étapes et choix de spécification et d'implantation du flot sont détaillés dans ce chapitre.

5.1. Rappel de la problématique

Pour améliorer la productivité de développement du code de configuration, nous cherchons à automatiser le flot que nous avons défini dans le chapitre précédent.

Dans ce contexte, nous pensons qu'une solution basée sur le standard IEEE 1685 IP-XACT [Ip-xact] peut répondre d'une manière efficace à nos besoins. Les raisons de ce choix sont nombreuses. En effet, ce standard est adapté à la représentation des architectures matérielles ciblées et en particulier les composants IPs. En effet, il permet la description de l'ensemble des ressources internes à savoir les registres et leurs adresses. De plus, il offre un mécanisme ouvert pour l'ajout d'informations non prévues par la norme.

Le standard IP-XACT offre également une interface de programmation (ou API) permettant d'accéder aux différentes informations du modèle. Il fournit un mécanisme d'automatisation des flots de génération de code à travers des composants d'intégration et de séquencement de différents outils. Enfin, plusieurs environnements de développement industriels, liés à ce standard, existent.

Dans ce chapitre, nous introduisons le standard IEEE 1685 (IP-XACT). Nous présentons ses concepts de base ainsi que les principales étapes du flot de conception qui lui sont associées. Nous expliquons par la suite comment nous avons spécifié notre flot de génération de code de configuration en se basant sur le flot proposé par IP-XACT. Nous abordons enfin l'intégration de notre solution basée sur IP-XACT dans l'environnement industriel de la société MDS (Magillem Design Services) [Magillem].

5.2. Présentation générale du standard IEEE 1685 (IP-XACT)

IP-XACT est un langage de description structurelle des architectures matérielles dont la version 1.5 a été standardisée en mars 2010. Ce standard permet principalement la documentation électronique des architectures à de bas niveaux d'abstraction, et en particulier au niveau RTL (Register Transfert Level) où les transferts de données sont décrits entre registres via des signaux. Les dernières versions (à partir de la version 1.4 de Mars 2008) offrent un support à la modélisation au niveau TLM (Transaction Level Model), via la définition des ports transactionnels. Le niveau TLM ne définit pas un modèle de représentation. Il s'agit d'une bibliothèque qui abstrait les détails de communication à travers des requêtes transactionnelles.

Le standard IP-XACT propose un flot de conception constitué de trois parties principales (Figure.5.1) : (1) une partie pour la construction d'une bibliothèque de composants matériels, (2) une pour l'assemblage des instances de ces composants pour construire les modèles d'architecture (Import et export des composants) et (3) une pour l'intégration et

l'enchaînement des outils externes. Elle est définie par des éléments IP-XACT de type « generator » et « generatorChain ». Les « generator » extraient les données des modèles d'architectures à travers une API fournie par le standard et nommée TGI (Tight Generator Interface). Elle est proposée sous forme de fichier wsdl (Web Service Description Language).

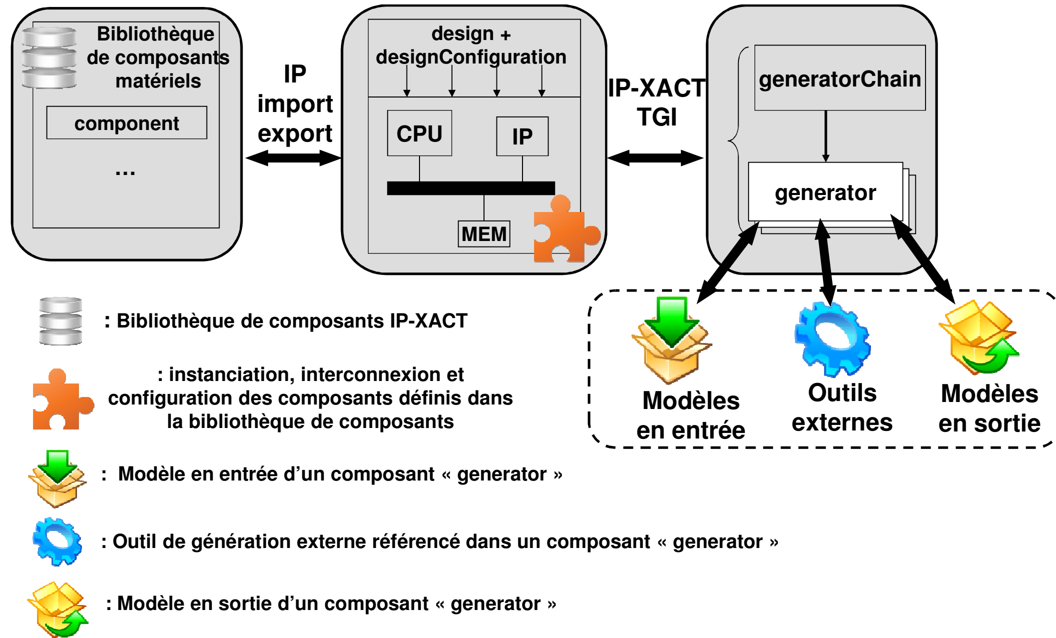


Figure 5.1. Les différentes étapes d'un flot de conception IP-XACT

5.2.1. Les composants de base du standard IEEE 1685

IP-XACT est basé sur le langage de balise XML et il est défini en 22 schémas XSD. Chacun est utilisé pour définir un objet particulier (composant, interface de communication, espace d'adressage, port, etc.). Ces schémas sont interdépendants et font appels les uns aux autres. La figure 5.2 illustre les dépendances entre les 9 principaux schémas d'IP-XACT.

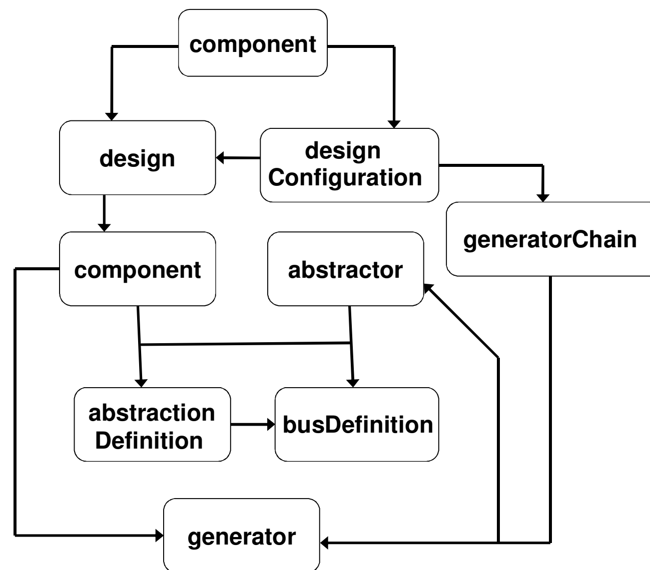


Figure 5.2. Dépendances entre les éléments de base du standard IP-XACT

Les composants matériels (IP, processeur, DSP, co-processeur, DMA, Timer, composants mémoires, bus, réseaux sur puce, cross-bar, etc.) qui forment la bibliothèque de composants IP-XACT sont décrits par l'élément « component » qui est l'élément central du standard (Figure.5.3). Un « busInterface » modélise l'interface de communication du « component » et référence un élément de type « abstractionDefinition ». Un « component » peut disposer de plusieurs vues qui référencent plusieurs représentations du même composant. Des fichiers liés au « component » (code source matériel, modèle de déploiement, logiciel embarqué, etc.) peuvent être associés à ce dernier.

Les éléments décrivant les interfaces de communication (Figure.5.3) sont :

- L'élément « busDefinition » décrit des aspects de haut niveau d'une interface de communication. Il permet de préciser les protocoles de communication (communication par adressage ou directe).
- L'élément « abstractionDefinition » décrit des aspects détaillés d'une interface de communication et référence un composant de type « busDefinition ». Il contient des informations précises sur le nombre de ports, leurs noms, leurs adresses, etc.

L'élément « design » instancie les « component » et décrit leurs interconnexions (Figure.5.3) en liant les ports décrits dans les « busInterface » de chaque composant.

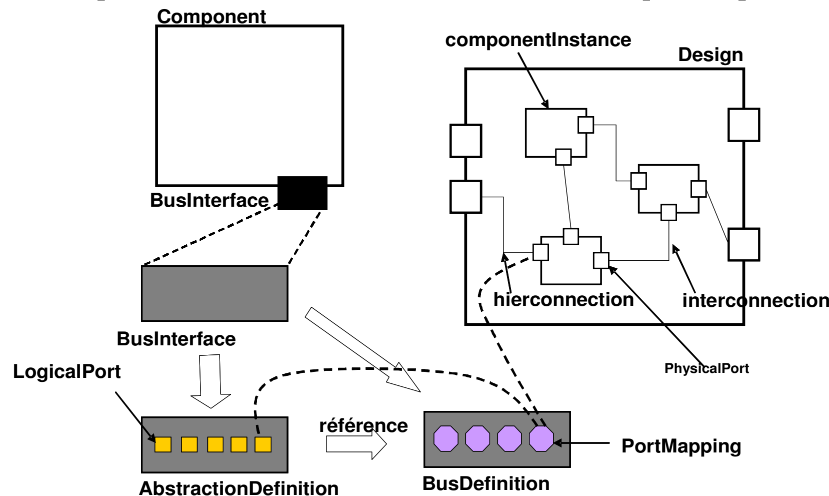


Figure 5.3. Composants de base du standard IP-XACT

L'élément « designConfiguration » joue le rôle d'un filtre qui extrait la partie adéquate des informations disponibles de la plateforme. Il peut préciser, par exemple, la vue du composant matériel que nous souhaitons utiliser, parmi celles qui sont définies dans la bibliothèque de composants. Une vue peut correspondre par exemple à un niveau d'abstraction.

Exemple : Dans la figure 5.4, un composant IP1 dispose de deux vues (une RTL et une TLM). Chacune référence un élément « component » (vue-TLM et vue-RTL) qui détaille son modèle d'architecture. Quand IP1 est instancié dans un « design », nous précisions la vue à prendre en considération à travers un élément « designConfiguration », que nous associons au

« desgin ». Il spécifie dans la figure 5.4 (lignes 2..3) que pour IP1, nous considérons la vue « vue-TLM ».

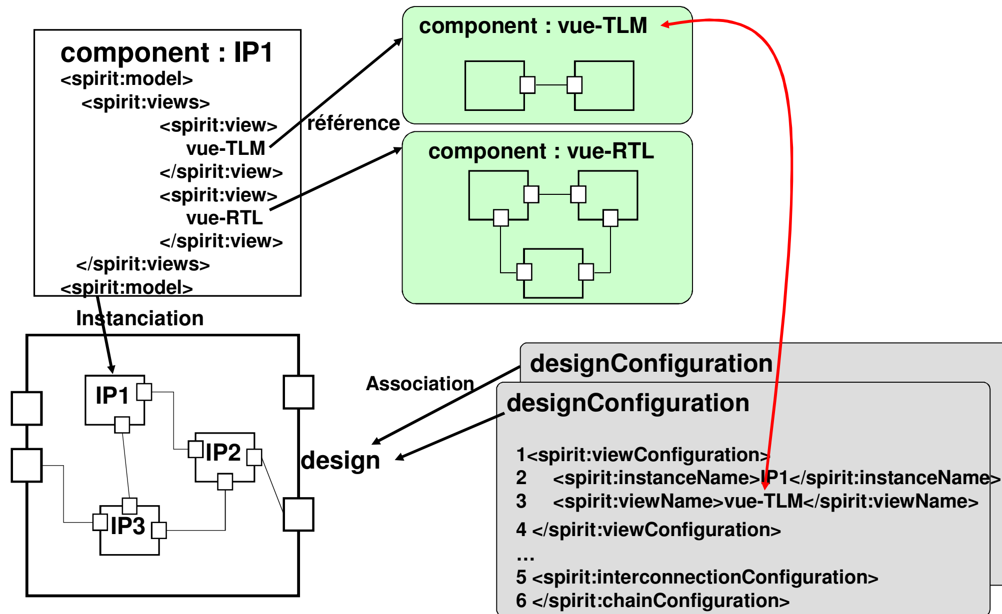


Figure 5.4. L'élément « designCofiguration» dans le standard IP-XACT

Chaque composant décrit en IP-XACT dispose d'un identificateur unique dans l'en-tête. Le standard IP-XACT fournit un mécanisme d'extension du langage à travers l'élément « vendorExtenions » qui peut être associé aux principaux éléments du langage. Il permet d'étendre la syntaxe des composants IP-XACT pour représenter des informations additionnelles et spécifiques à un fournisseur d'IPs particulier.

5.2.2. Intégration et enchaînement des outils dans IP-XACT

L'intégration et l'enchaînement des outils sont réalisés grâce aux éléments IP-XACT de type « generator » et « generatorChain ».

Le premier (Figure.5.5.a) décrit un outil (de simulation, compilation, génération de code, etc.) sous format XML. Il référence son exécutable (balise « spirit :generatorExe ») et initialise ses paramètres d'entrée et de sortie (balise « spirit :parameter »). Il extrait les données depuis les « design » IP-XACT via les primitives de lecture et d'écriture du TGI.

L'élément « generatorChain » (Figure.5.5.b) permet la description des enchaînements des outils à travers le séquencement des éléments de type « generator ».

Un flot de génération de code est constitué généralement d'un ensemble de modèles en entrée, d'outils (de génération de code, de transformation de modèles, de simulation, etc.) et de modèles en sortie. Un modèle en entrée peut décrire l'architecture, l'application ou le partitionnement des ressources. Il peut être également un modèle intermédiaire. Les fichiers constituant le code généré (code SystemC, configurations, microprogrammes, etc.) constituent les sorties du flot.

Pour réaliser l'intégration d'un flot de génération de code dans un environnement IP-XACT, il faudrait décomposer ce flot en un ensemble d'éléments « generator » séquencés à travers un « generatorChain ». Les modèles en entrée et en sortie ainsi que les modèles intermédiaires sont décrits comme les paramètres des « generator ». Les exécutables des outils de transformation et de génération sont référencés également par les éléments « generator ».

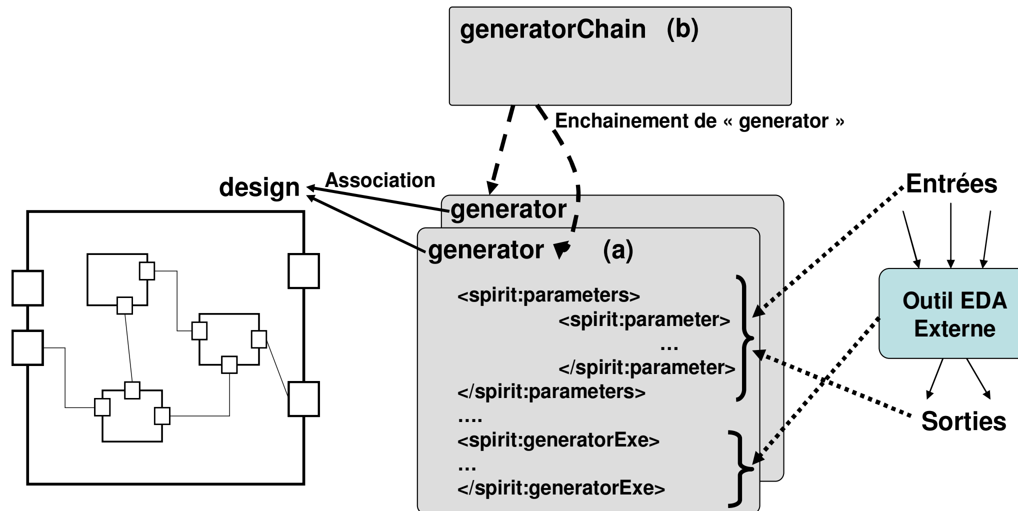


Figure 5.5. Les éléments « generator » et « generatorChain » dans le standard IP-XACT

5.2.3. Outils de développement IP-XACT

Des outils et des environnements industriels comme Magillem [Magillem], Nauet [Nauet], plugin Eclipse développé par ARM [Plugin], Scarlet [Scarlet], Duolog [Duolog] ont été développés afin de faciliter l'utilisation du standard IP-XACT, en fournissant des fonctionnalités de conception et d'intégration des outils et des composants matériels.

D'autres travaux autour de la représentation du langage IP-XACT avec UML ont été élaborés dans [And08] et [Rev08] à travers la proposition de profils UML. Leur objectif principal est d'utiliser le langage graphique UML, à travers des stéréotypes IP-XACT, pour décrire des architectures matérielles. Ainsi, les utilisateurs peuvent bénéficier des fonctionnalités des outils déjà développés autour de ce langage graphique (éditeurs graphiques, outils de transformations, etc.) pour exploiter des modèles IP-XACT.

Nous ne cherchons pas dans cette thèse à représenter IP-XACT sous un autre format. Nous souhaitons spécifier et implanter notre flot de génération de code de configuration dans un environnement IP-XACT. Ensuite, nous cherchons à l'intégrer dans un outil industriel. A la fin de ce chapitre, nous présentons l'outil industriel Magillem dans lequel nous intégrons notre flot en respectant les différentes étapes du flot IP-XACT.

5.3. Spécification du flot de génération de code de configuration basé sur le flot IP-XACT

Dans cette section, nous décrivons les phases de spécification de notre flot de génération de code de configuration en utilisant le flot IP-XACT (Figure.5.6).

5.3.1. Choix du standard IP-XACT

Nous avons choisi le standard IP-XACT comme base de cette partie de nos travaux, pour différentes raisons:

1) Nous pouvons profiter de la réutilisation des outils existants qui gèrent cette norme et qui fournissent une implantation de l'API « TGI ». Nous rappelons que cette dernière permet l'extraction des données depuis les éléments « design ». Ces outils fournissent des IDE (Integrated Development Environment) graphiques pour modéliser les architectures et pour développer des générateurs. En plus, l'utilisation de l'API « TGI » dans le développement des générateurs favorise leur portabilité dans différents environnements IP-XACT.
2) Nous pouvons exploiter les mécanismes d'automatisation des flots fournis par le standard IP-XACT à travers les éléments "generator" et "generatorChain". Il s'agit d'assembler un ensemble d'outils développés séparément dans une chaîne d'outils. Ainsi, IP-XACT permet d'exporter non seulement les modèles d'architecture, mais aussi les chaines d'outils correspondants.
3) Le standard IP-XACT fournit un mécanisme d'extension simple à utiliser qui permet d'ajouter des informations non prévues par la norme. Cela permet d'étendre la syntaxe d'IP-XACT afin de représenter n'importe quelles informations additionnelles spécifiques à un fournisseur d'IPs. Ces informations supplémentaires peuvent être matérielles ou logicielles. Elles ne sont pas automatiquement interprétées par les outils standards (API TGI). Mais, il faut développer également les outils adéquats (extension de l'API TGI) supportant l'accès à ces informations spécifiques.
4) Le standard est adapté à la représentation de la structure des architectures matérielles que nous ciblons à savoir les architectures à base d'IPs. Il a été défini pour décrire les IPs dans l'objectif de leur réutilisation et leur intégration dans différents environnements de développement.

5.3.2. IP-XACT pour le flot de génération de code de configuration

En premier lieu, nous cherchons à décrire, avec le standard IP-XACT, les architectures matérielles à base d'IPs et d'interfaces réseau configurables. Dans le flot IP-XACT, cela revient à décrire les différents composants matériels (les composants IPs, les interfaces réseau, les périphériques de contrôle, etc.) dans une bibliothèque de composants.

Ces composants sont par la suite instanciés et interconnectés dans un élément « design » pour définir la structure de la plateforme.

Pour représenter l'aspect configurable des plateformes que nous ciblons, nous avons ressenti le besoin d'étendre le standard IP-XACT au niveau des éléments « component », « design » et « designConfiguration ». Ces extensions nous permettent de décrire les chemins de communication physiques ainsi que le « HAL » de chaque IP. Elles permettent également de décrire les interfaces réseau configurables comme une association des interfaces réseau classiques avec les périphériques de contrôle.

Nous intitulons ARDL (ARchitecture Description Level) le modèle de représentation des plateformes utilisant IP-XACT avec les extensions que nous proposons. Nous détaillons ces extensions dans la prochaine section. D'autre part, nous appelons APDL (APplication Description Level) le modèle de représentation des applications orientées flot de données.

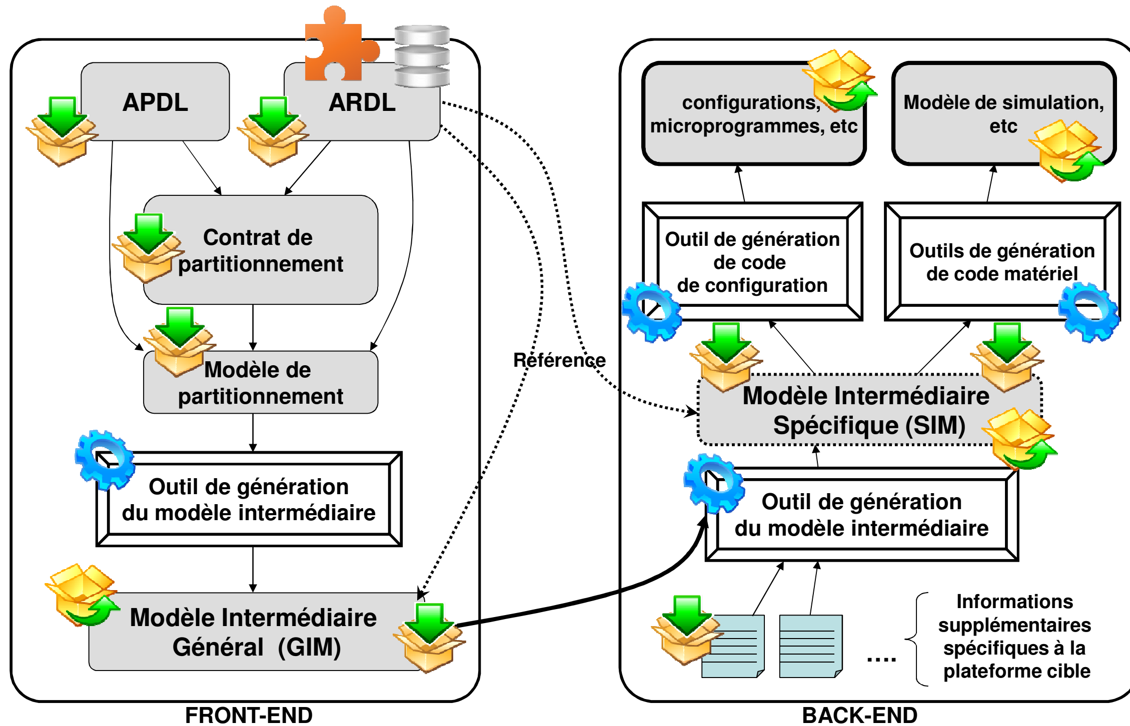


Figure 5.6. Spécification du flot de génération de code de configuration en respectant les phases du flot IP-XACT

Les représentations ARDL et APDL, ainsi que le contrat de partitionnement et le modèle de partitionnement sont les entrées de l'outil de génération du modèle « GIM ». Une fois généré, le « GIM » est référencé par le modèle ARDL de départ.

Dans la partie « Back-End » du flot, un nouveau élément IP-XACT « generator » permet la génération du second modèle intermédiaire « SIM ». Il dispose de deux entrées : le modèle « GIM » et un fichier contenant des informations spécifiques à la plateforme et nécessaires à la génération du code de configuration. Ces informations sont encapsulées dans un élément IP-XACT « designConfiguration ». Pour les représenter, nous avons apporté des extensions à ce dernier, que nous détaillons dans la suite.

Une fois généré, le « SIM » est également référencé par le modèle ARDL de départ. D'autres éléments « generator » sont définis pour la génération du code de configuration final. Tous ces éléments prennent le modèle « SIM » comme un modèle d'entrée et référencent les outils de génération de code (qui sont un outil de génération de code de configuration, un outil de génération de code de simulation, etc.).

Dans la suite du chapitre, nous détaillons les étapes de spécification du flot de génération de code de configuration conformément au flot IP-XACT. Nous décrivons dans la section suivante les extensions que nous avons apportées au standard IP-XACT pour pouvoir représenter les plateformes en ARDL.

La section 5 s'intéresse à la représentation APDL et des modèles de partitionnement. Nous abordons dans la section 6 la mise en œuvre des mécanismes d'automatisation du flot de génération de code de configuration. Nous détaillons également les modèles intermédiaires. Nous clôturons ce chapitre par la section 7 où nous exposons la manière dont nous avons intégré notre flot dans un environnement IP-XACT industriel nommé Magillem.

5.4. Extension du standard IP-XACT pour le modèle de représentation ARDL

Dance cette section, nous décrivons notre approche de mise en œuvre du modèle de représentation ARDL pour les plateformes à interfaces réseau configurables, à travers les extensions apportées au standard IP-XACT. Nous rappelons qu'IP-XACT ne permet pas d'exprimer les chemins de communication physiques ainsi que les fonctionnalités fournies par les IPs matériels. Les extensions, que nous avons rajoutées, essayent de palier ces limitations.

5.4.1. Extension de l'élément « component » pour la description des fonctionnalités des composants matériels

Nous cherchons à représenter à haut niveau les composants des plateformes à base d'IPs (interface réseau configurable, IPs, processeur d'orchestration, périphérique de contrôle, etc.). Nous souhaitons également représenter les caractéristiques matérielles et les fonctionnalités fournies par les composants de l'architecture. Le standard IP-XACT ne permet pas de représenter ces fonctionnalités. En ce qui concerne la description à haut niveau, nous rappelons qu'IP-XACT cible la description des IPs en RTL. Pour représenter par exemple un processeur, le standard exige l'utilisation de l'élément « spirit :cpu ». Or, l'utilisation de cet élément nous oblige à décrire les espaces d'adressage associés à ce composant. La description du processeur d'orchestration n'exige pas la présence de détails tels que les espaces d'adressage. Ainsi, nous apportons une extension à l'élément « component » d'IP-XACT pour répondre à nos besoins. L'extension se compose de deux parties principales :

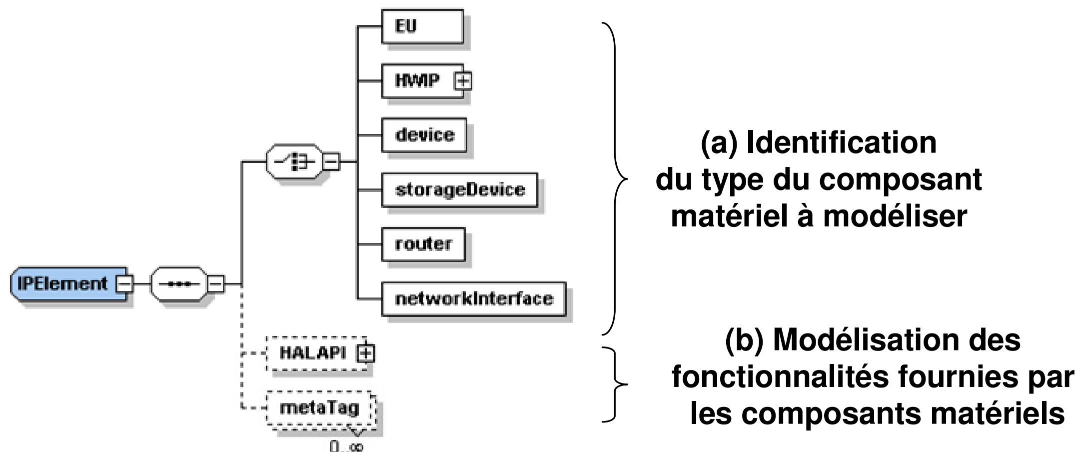


Figure 5.7. Shéma XSD représentant l'extension du standard IP-XACT pour le modèle de représentation ARDL (vue graphique du logiciel XMLSpy [Altova])

(1) La première partie permet d'identifier le type de composant à modéliser (Figure.5.7.a). Nous réutilisons les interfaces de communication transactionnelles du standard IP-XACT, pour chaque composant. Grâce à l'extension, nous pouvons décrire notamment, des processeurs (EU : Execution Unit) dont le rôle est d'orchestrer l'exécution du code de configuration. Elle nous permet de modéliser également les composants matériels tels que les IPs (HWIP) sur lesquels les tâches applicatives seront réparties, les périphériques de contrôle (device) ainsi que les composants de mémorisation (storageDevice). Nous pouvons également représenter les composants de communication comme les routeurs (router) et les interfaces réseau (networkInterface).

(2) La deuxième partie permet de décrire les fonctionnalités fournies par un composant matériel. Elle concerne la balise « HALAPI » (Figure.5.7.b). Celle-ci comporte plusieurs balises de type « HALPrimitive » (Figure. 5.8). Une fonctionnalité peut être décrite de deux manières différentes. La première contient un ensemble de paramètres (Figure.5.8.a). Elle est associée aux composants IPs qui fournissent plusieurs fonctionnalités nécessitant chacune une configuration particulière des registres de l'IP. La deuxième manière consiste à représenter la fonctionnalité avec une signature (Figure.5.8.b). Cette dernière contient le nom de la fonction, la liste de ses arguments et le type de la valeur de retour.

Le contenu de la balise « HALAPI » correspond à la couche « Abstraction du matériel » de la structure en couches du code de configuration, présentée au paragraphe 4.2.1.

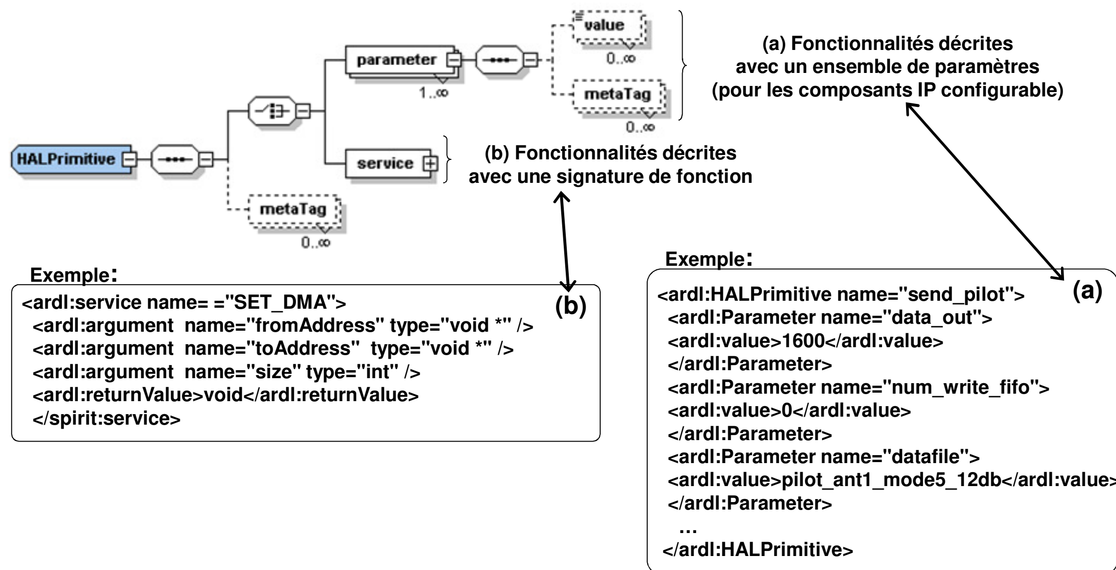


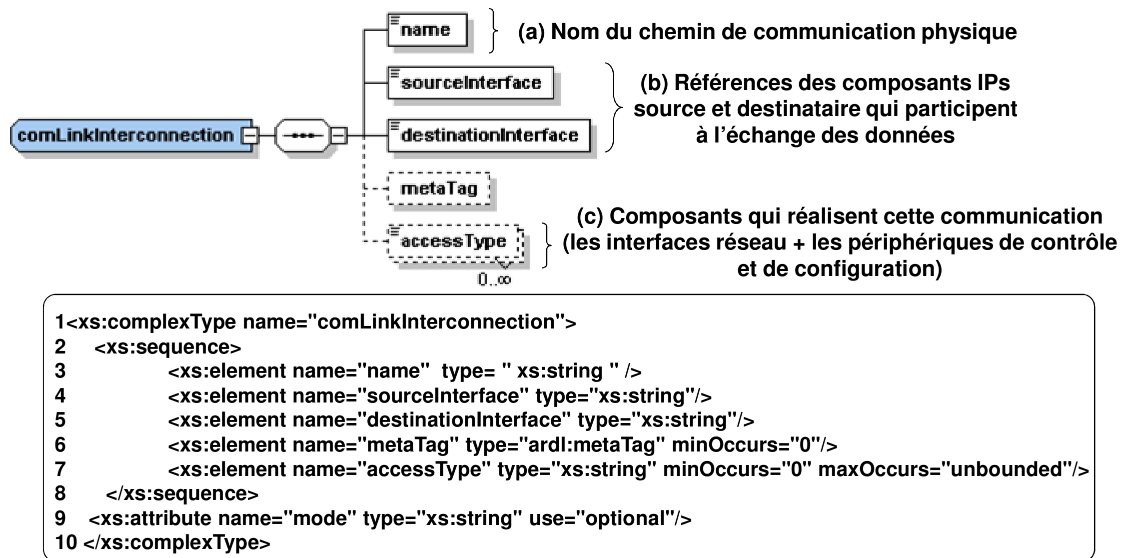
Figure 5.8. Extension du standard IP-XACT pour la représentation de l'API d'abstraction du matériel

5.4.2. Extension de l'élément « design » pour la description des chemins de communication physiques

Nous cherchons à représenter les chemins physiques de communication exploitables dans une plateforme matérielle. En IP-XACT, cette dernière est modélisée par un élément « design ». Or, un élément « design » permet uniquement l'instanciation et l'interconnexion des composants, sans mentionner les chemins de communication de la plateforme qui peuvent être utilisés.

Ainsi, l'extension que nous avons rajoutée à l'élément « design » d'IP-XACT a pour but d'exprimer les chemins de communication physiques entre les blocs IPs du système. Il s'agit de définir les chemins autorisés ou interdits par l'architecture. Cette extension permet également de donner des précisions sur l'emplacement des instances des composants IP sur le réseau et par rapport aux routeurs.

Pour représenter les chemins de communication physiques, nous définissons une structure nommée « comLinkInterconnection » (Figure.5.9). Un chemin de communication physique est défini par un nom (Figure.5.9.a), un composant IP source, un composant IP destinataire (Figure.5.9.b) ainsi que les composants matériels qui interviennent pour réaliser cette communication (Figure.5.9.c) à savoir, les interfaces réseau et les périphériques de communication (périphérique DMA par exemple).




```
1<xs:complexType name="comLinkInterconnection">
2   <xs:sequence>
3           <xs:element name="name"  type= " xs:string " />
4           <xs:element name="sourceInterface" type="xs:string"/>
5           <xs:element name="destinationInterface" type="xs:string"/>
6           <xs:element name="metaTag" type="ardl:metaTag" minOccurs="0"/>
7           <xs:element name="accessType" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
8    </xs:sequence>
9   <xs:attribute name="mode" type="xs:string" use="optional"/>
10 </xs:complexType>
```

Figure 5.9. Description d'un chemin de communication physique

Pour positionner les IPs sur le réseau sur puce, nous avons besoin, lors de l'instanciation de ces composants, d'ajouter des informations concernant leurs emplacements par rapport au réseau (attachement du composant IP à un routeur du réseau) et par rapport au routeur (direction de l'emplacement de l'IP par rapport au routeur du réseau : Sud, Nord, Est, Ouest, Ressource). Ces informations peuvent être extraites à partir du schéma des interconnexions entre les composants matériels de l'architecture. Nous préférons, cependant, les décrire d'une manière explicite pour faciliter leurs traitements par les outils de génération de code de configuration. Ces informations doivent être revues et mises à jour lors de la réutilisation des IPs.

5.4.3. Extension de l'élément « designConfiguration » pour la représentation des informations spécifiques à la plateforme

Nous cherchons à représenter les informations spécifiques à la plateforme cible. Elles permettent le passage d'une représentation de haut niveau à une représentation plus détaillée de l'architecture (informations concernant le plan mémoire des différents composants, les pilotes à associer aux composants matériels, les chemins de communication à exploiter, la

grammaire des configurations des composants, etc.). C'est le cas du passage du modèle « GIM » vers le modèle « SIM ». Le standard IP-XACT ne fournit pas d'éléments qui représentent ces informations additionnelles. Par contre, il définit l'élément « designConfiguration » qui joue le rôle d'un filtre et extrait une partie des informations disponibles d'un « design ».

Nous pensons qu'une évolution du « designConfiguration » peut être élaborée afin d'étendre ces fonctionnalités de filtre. Ainsi, il peut apporter des détails supplémentaires au « design » et ne se limite pas à traiter et à filtrer les informations déjà existantes.

Les informations que nous ajoutons à un élément de type « design » décrivant une architecture à base d'IPs, peuvent être classées en deux types : Celles concernant les chemins de communication et celles qui représentant les informations spécifiques à la plateforme ciblé.

5.4.3.1. Complément des chemins de communication

Si les chemins de communications physiques ne sont pas décrits au niveau du « design », on peut compléter leur description en rajoutant au « design » un élément « desginConfiguration » contenant ces informations. On peut éventuellement créer plusieurs « designConfiguration » qui définissent différents schémas de chemins de communication possibles, et ceci dans le but de tester divers modèles de partitionnement. La description d'un chemin de communication physique dans l'extension de l'élément « designConfiguration » est la même que celle décrite dans la figure 5.9.

5.4.3.2. Ajout des informations de raffinement

Nous avons également étendu l'élément « designConfiguration » pour faciliter le raffinement du modèle « GIM » vers le modèle « SIM ». Nous avons structuré ces extensions en trois éléments :

« simComponentConfiguration » : La représentation de la plateforme dans le modèle GIM est constituée de ressources matérielles auxquelles sont associées des ressources applicatives. Cette extension permet d'associer à chacun de ces composants du GIM un composant d'une bibliothèque (SystemC, SocLib par exemple). Cela permet la génération du code du modèle de simulation. Supposons qu'on dispose d'un périphérique « timer » dans le modèle « GIM ». Nous lui associons le composant « VciTimer » de la bibliothèque SoCLib pour générer le modèle de simulation. Cette affectation est décrite avec l'élément « simComponentConfiguration ».

Nous associons également aux composants configurables, la grammaire et la structure des configurations exécutables sur la plateforme. Cela permet la génération du code de configuration. Supposons qu'on dispose d'un périphérique de contrôle « OCC » dans le modèle « GIM ». Pour pouvoir programmer ce composant, nous avons besoin de connaitre la syntaxe des configurations à développer. Cette dernière est fournie par l'élément « simComponentConfiguration ». Il spécifie qu'une configuration « OCC » est décrite par les paramètres « num_icc », « path_to_target », « sel_credit », etc.

« simSegmentConfiguration » : La description du plan de répartition de la mémoire de l'architecture matérielle permet de définir plusieurs distributions de la mémoire pour un

même modèle « GIM ». Il s'agit d'associer les segments mémoire à l'ensemble des composants. Cela est utile pour générer le logiciel binaire exécutable pour les processeurs.

« simComLinkConfiguration » : Cette extension est utilisée pour associer des pilotes de périphériques aux chemins de communication physiques. Il peut s'agir d'une association entre un chemin de communication contenant un composant DMA avec le pilote de ce dernier, par exemple. Les détails sur d'implantation de ces extensions sont présentés dans l'annexe A.

5.5. Spécification du modèle de représentation de l'application APDL et des modèles de partitionnement

5.5.1. Modèle de représentation des applications orientées flot de données : APDL

Nous avons établi un modèle de représentation des applications orientées flots de données que nous avons appelé APDL. Il ne s'agit pas d'un nouveau langage, mais d'une représentation textuelle en XML, décrivant l'enchaînement du flot de données applicatif. Nous avons choisi de le développer en XML pour qu'il soit conforme avec le langage IP-XACT et la représentation ARDL.

Dans cette représentation, une application (Figure. 5.10) est constituée par des éléments « tasks » listant les tâches applicatives, « channels » pour énumérer les canaux de communication, « interconnections » pour lister les interconnexions entre les ports de tâches et les ports de canaux et « variables » pour énumérer les variables à utiliser pour décrire des flots de données dynamiques. Ces variables sont par la suite utilisées pour la description des arbitres dans les canaux de communication. L'élément « tasks » est constitué d'un ou plusieurs éléments « task » ; et l'élément « channels » est constitué également d'un ou plusieurs éléments « channel ».

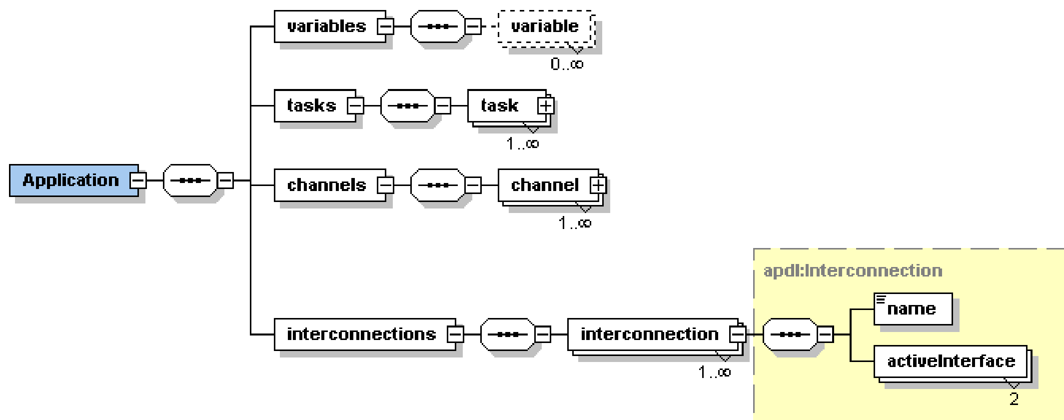


Figure 5.10. Vue graphique du schéma XSD de la représentation APDL

Les interconnexions sont décrites de la même manière que dans le langage IP-XACT. Nous avons fait ce choix pour faciliter l'intégration de notre représentation dans un environnement IP-XACT. Ainsi, une interconnexion est définie par un nom et par deux éléments « activeInterface ». Chacun référence le composant à interconnecter (une tâche ou

un canal), en plus de l'interface de communication, à savoir les ports de tâches et/ou de canaux de communication logiciels.

Un élément « task » décrit une tâche applicative et dispose d'un attribut qui référence son fichier source. Nous pouvons associer à un élément « task » plusieurs ports de tâches qui sont interconnectés avec des ports de canaux de communication logiciels. Les ports de tâches et de canaux de communication sont décrits avec une même structure qui spécifie le nom du port, sa direction et la taille de la structure des données transitant à travers le port.

Un élément « channel » dispose de plusieurs attributs, à savoir le type du canal (FIFO ou mémoire partagée), sa profondeur, etc. Le canal de communication logiciel est constitué de deux éléments « channelAccess », un pour l'entrée du canal et un pour la sortie. Un « channelAccess » liste les ports du canal et décrit le gestionnaire d'accès en entrée ou en sortie (arbitre).

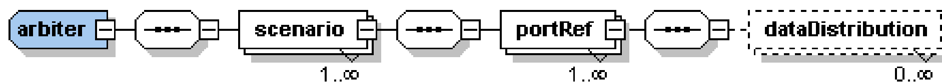


Figure 5.11. Description d'un arbitre pour les canaux de communication logiciels

Un arbitre permet d'ordonnancer les entrées et les sorties des données à travers les canaux de communication. Il est optionnel (Figure. 5.11) et permet de modéliser des applications KPN par exemple. Il suffit de créer des « channelAccess » avec un unique port d'entrée et de sortie, sans arbitre des deux cotés.

Un arbitre est constitué par un ensemble de scénarii. Chacun dispose d'un attribut «occurrence» qui représente le dernier élément d'une colonne dans une matrice d'entrée ou de sortie. Ces matrices ont été introduites en 4.5.2. Il s'agit de la valeur qui correspond au nombre de fois où ce scénario est exécuté. Elle peut prendre la valeur d'une « variable » déclarée dans le but de décrire des flots de données dynamiques. Un scénario est constitué par une ou plusieurs éléments «portRef».

Un « portRef » référence un port du canal, interconnecté à son tour à un port d'une tâche. Cet élément dispose d'un attribut « portrefOccurence » qui détermine le nombre de lectures ou d'écritures depuis/vers ce port. Cet attribut correspond aux valeurs qui constituent une colonne dans la matrice d'entrée ou de sortie. Enfin, l'élément « dataDistribution », qui est optionnel, permet de décrire explicitement la distribution des données entrantes ou sortantes d'un port de canal vers les tâches destinataires. Des détails d'implantation sont présentés dans l'annexe B.

5.5.2. Modèles de partitionnement

Nous présentons dans ce paragraphe la manière dont nous exprimons les règles de partitionnement que nous souhaitons appliquer lors du déploiement des ressources. Ces règles sont définies dans le contrat de partitionnement.

Le contrat de partitionnement est développé en XML (Figure 5.12). Ce dernier comporte deux types de contraintes :

- Les contraintes générales (Figure. 5.12.A) associent à chaque type de composants matériels (IPs, interfaces réseau, périphériques de contrôle, chemins de

communication physiques, etc.) les types de ressources applicatives (tâches applicatives, canal de communication logiciel, arbitre, etc.) pouvant être déployées dessus.

- Les contraintes spécifiques (Figure. 5.12.B) sont liées aux instances des ressources d'architecture et de l'application. Nous pouvons par exemple exprimer l'ensemble des tâches applicatives bien définies qui peuvent s'exécuter, sur un IP bien déterminé.

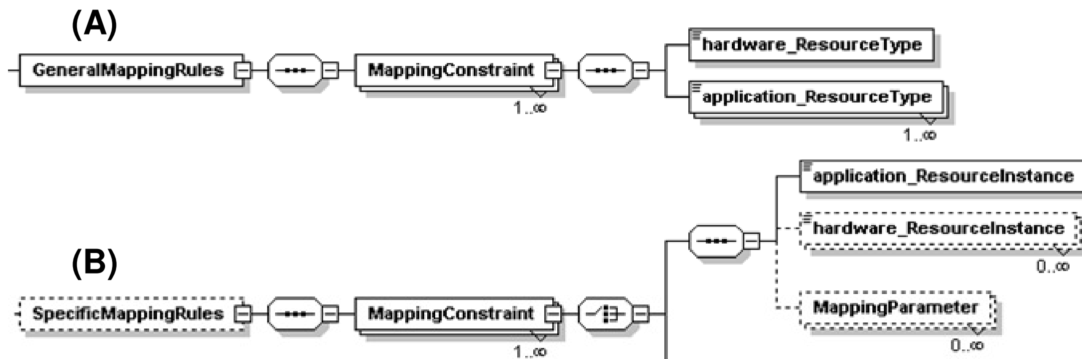


Figure 5.12. Schéma XSD des contraintes de partitionnement générales et spécifiques dans le contrat de partitionnement

Le modèle de partitionnement décrit le déploiement de l'application sur l'architecture matérielle, en respectant les règles précédemment définies dans le contrat de partitionnement.

Le partitionnement dans notre flot est réalisé manuellement et il est à la charge d'une personne indépendante ou non de celles qui ont modélisé l'application et l'architecture. Il est décrit également en XML. Les règles de partitionnement spécialisent la syntaxe générale de partitionnement pour n'utiliser que la partie conforme à l'ensemble des contraintes.

Dans le modèle de partitionnement, les tâches applicatives sont associées aux composants de calcul de type IP. Nous mettons en correspondance les ports de tâches, dans le cas où elles contiennent des informations de flux de données, avec les interfaces réseau ou les périphériques de contrôle.

Les canaux de communication logiciels sont associés aux chemins de communication physiques ou aux composants IPs si ces derniers fournissent les fonctionnalités de communication (Exemple : composant mémoire programmable). Les ports de canaux et les arbitres sont associés, à leur tour, aux interfaces réseau et aux périphériques de contrôle. Des exemples d'utilisation des modèles de partitionnement sont présentés dans le chapitre 6. Des détails d'implantation sont présentés dans l'annexe C.

5.6. Automatisation du flot de génération de code de configuration

L'automatisation du flot de génération du code de configuration est réalisée à partir des représentations de haut niveau de l'application, de la plateforme et du partitionnement. Ces dernières sont définies d'une manière manuelle.

L'automatisation de la génération de code est spécifiée dans la troisième et dernière phase du flot IP-XACT. Il s'agit d'une étape d'enchaînement des outils de génération et de transformation. Pour réaliser cette étape, nous avons développé des composants IP-XACT de type « generator » et « generatorChain ».

5.6.1. Les composants « generator » et « generatorChain » pour l'automatisation du flot de génération de code de configuration

Nous réalisons l'automatisation de notre flot de génération de code de configuration, grâce à la définition de composants IP-XACT de type « generator » et « generatorChain ». La génération de chacun des deux modèles intermédiaires « GIM » et « SIM » est réalisée à travers un composant IP-XACT « generator ».

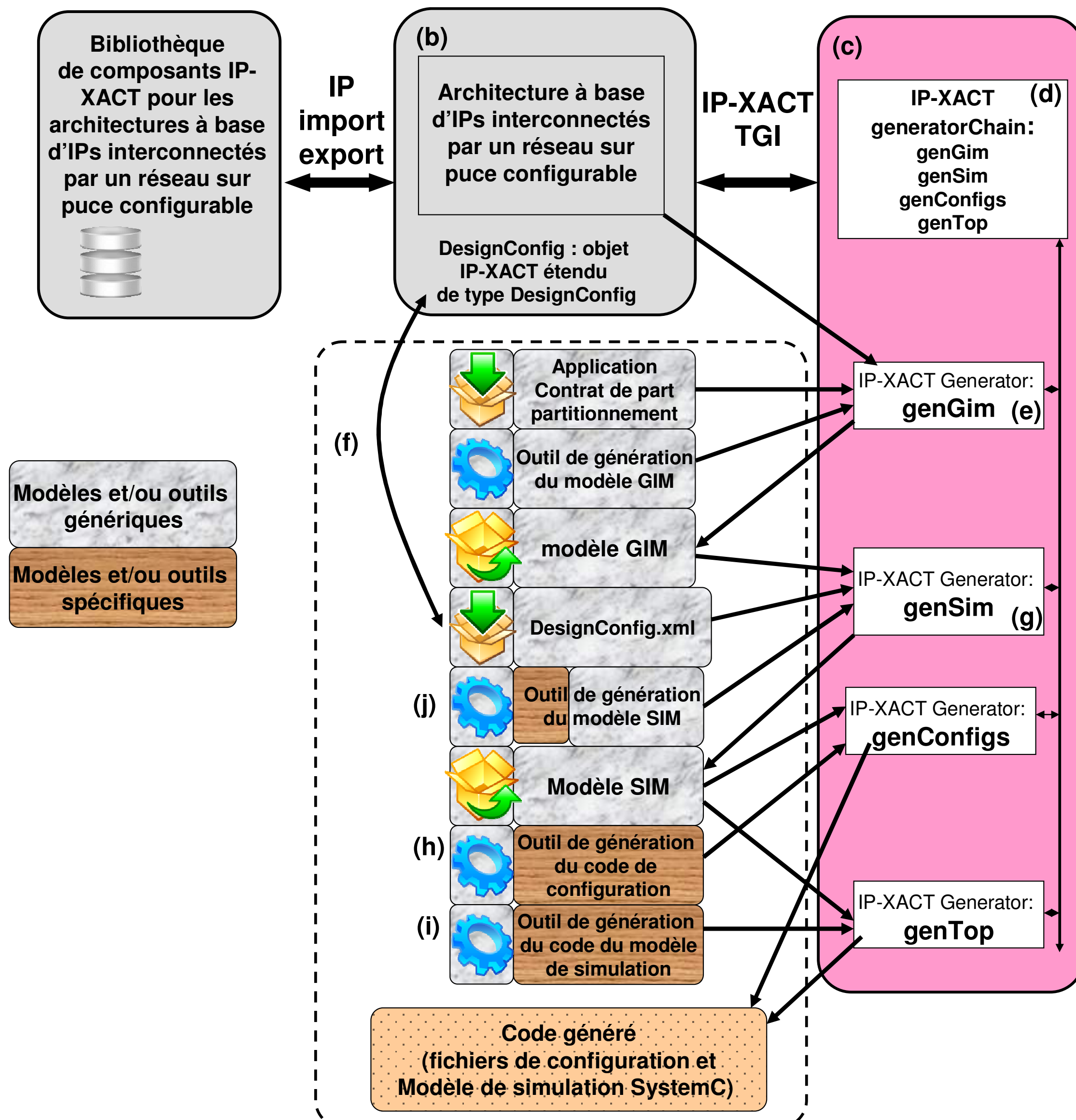


Figure 5.13. Automatisation du flot de génération de code de configuration dans un environnement IP-XACT

En premier lieu, nous définissons un « generator » nommé « genGim » (Figures 5.13.e). Il est utilisé pour la génération du modèle « GIM ». Il prend comme paramètres les modèles APDL, ARDL, de partitionnement ainsi que l'exécutable de l'outil de génération. Il s'agit d'un outil générique puisqu'il est indépendant des caractéristiques de la plateforme cible. En effet, nous ne traitons pas encore les caractéristiques de la plateforme à ce niveau du flot.

En deuxième lieu, nous développons un second « generator » nommé « genSim » (Figure. 5.13.g). Ce dernier est utilisé pour la génération du modèle « SIM ». Il prend comme paramètres d'entrée le modèle « GIM » ainsi que les informations supplémentaires et spécifiques à la plateforme cible. Ces informations sont décrites dans un élément « designConfiguration » étendu (voir 5.4.3) d'IP-XACT.

L'outil de génération du modèle SIM (Figure. 5.13.j) est un outil semi-générique. En effet, sa partie générique correspond à celle qui traite les informations provenant du modèle « GIM ». Sa partie spécifique sert au traitement des données provenant de l'élément « designConfig » étendu. Une fois généré, le modèle « SIM » est également référencé, par le modèle ARDL.

Nous définissons enfin les « generator » IP-XACT nommés « genConfigs » et « genTop » qui encapsulent les outils de génération de code final. Le composant « genConfigs » référence l'outil de génération de code de configuration (Figure. 5.13.h), qui est spécifique à la plateforme cible. Il prend comme paramètre d'entrée le modèle « SIM ».

Le composant « genTop » prend également en entrée le modèle « SIM » ainsi que l'outil de génération du code du modèle de simulation (Figure. 5.13.i). Cet outil est également spécifique à la plateforme cible et fait référence à des composants matériels bien définis dans des bibliothèques (SystemC, SoCLib, etc.). L'enchainement de tous ces « generator » (genGim, genSim, genConfigs, genTop) sont définis par un composant IP-XACT de type « generatorChain ».

5.6.2. Les modèles intermédiaires « GIM » et « SIM »

Le modèle GIM associe, dans un format XML et d'une manière structurelle, les tâches applicatives aux IP matériels. Il déploie également les informations relatives au trafic des données applicatives (les arbitres à l'entrée et à la sortie des canaux de communication, les ports des canaux de communication) sur les interfaces réseau ainsi que les périphériques de contrôle. Les canaux de communication logiciels sont associés aux chemins de communication physiques décrits dans le modèle ARDL.

Le modèle intermédiaire « SIM » est le résultat du raffinement du modèle « GIM ». Ce raffinement est réalisé en intégrant au modèle « GIM » de nouvelles informations relatives à la configuration de la plateforme cible, sa syntaxe, les composants matériels utilisés dans le modèle de simulation et l'affectation des espaces d'adressage aux composants de la plateforme. Les informations spécifiques à la plateforme sont décrites dans un composant « DesignConfig » étendu d'IP-XACT (voir 5.4.3).

Les informations incluses dans le modèle « SIM » peuvent contenir les configurations des composants de l'architecture ou des outils et des environnements de développement du logiciel ou/et du matériel (choix des bibliothèques de communication, des pilotes, des

systèmes d'exploitation etc.). Ces nouvelles informations sont représentées par les éléments suivants :

- « SimVirtualComponent » : permet d'associer des composants matériels faisant partie d'une bibliothèque de composants (SystemC, TLM, SoCLib, etc.) aux composants matériels décrits dans le modèle GIM. Cela permet la génération du modèle de simulation.
- « SimDriver » : permet d'affecter les pilotes à utiliser aux composants matériels décrits dans le modèle GIM.
- « SimConfiguration » : permet la description d'une configuration d'une manière générale. Les éléments « SimConfiguration » sont associées aux composants matériels de calcul (IP matériel) et de communication (interfaces réseau) ainsi qu'aux périphériques de contrôle. Une configuration dans le modèle « SIM » dispose d'un type et d'un ensemble de paramètres.
- « SimMemorySegment » : permet de décrire les affectations des espaces d'adressage aux composants matériels de la plateforme.

La figure 5.14 illustre un exemple de passage du modèle « GIM » vers le modèle « SIM » en utilisant les informations fournies par l'élément étendu d'IP-XACT « designConfiguration ». Le composant « occ0_mep_22 » dans le « GIM » contient les informations du port « mephisto22_out » de la tâche applicative « mephisto22 » (Figure 5.14.a). La figure 5.14.b illustre la syntaxe des configurations de type « OCC » à produire. Elle est décrite avec la balise « simComponentConfiguration ». Le « SIM » traite les données provenant de ces deux modèles pour fournir une vue orientée plateforme cible du composant « OCC ». Ce dernier contient une balise « simConfiguration » qui initialise les paramètres de sa configuration (sel_credit, num_icc, path_to_target, etc.) spécifique à la plateforme Magali.

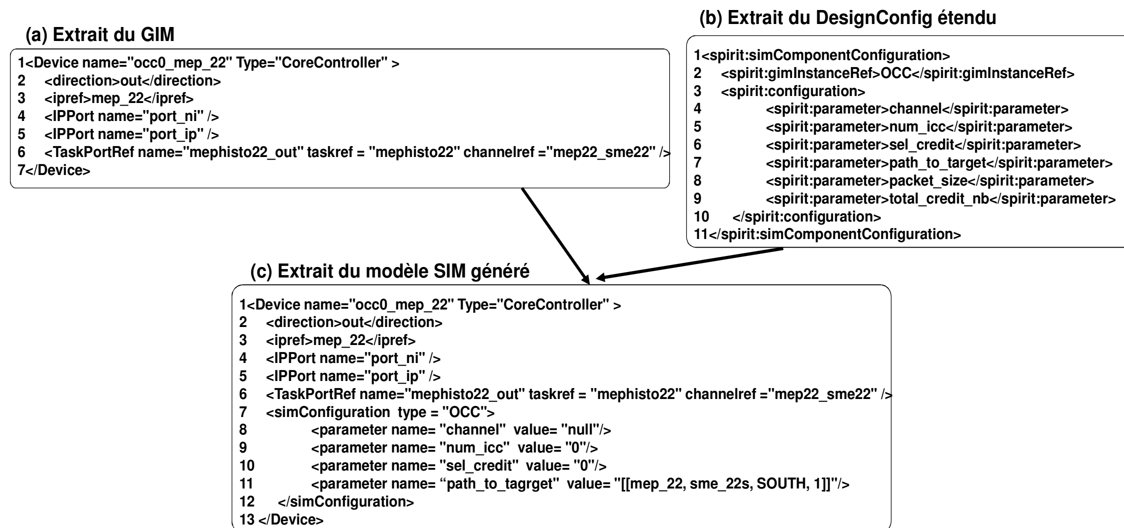


Figure 5.14. Exemple de génération d'une partie SIM depuis le GIM et les informations spécifiques à la plateforme

5.6.3. Synthèse

A ce stade, nous avons spécifié et implanté notre flot de génération de code de configuration pour les plateformes contenant des interfaces réseau configurables, à l'aide du standard IP-XACT. Pour cela, nous avons dû rajouter des extensions au standard, afin de pouvoir modéliser des composants et des informations non prévues par la norme. Nous avons développé plusieurs outils séparément (outils de génération du « GIM », du « SIM », etc.). Ils ont été ensuite assemblés dans une chaîne d'outils ce qui a permis l'automatisation du flot et la génération rapide du code de configuration.

Notre flot fonctionne indépendamment des environnements industriels d'IP-XACT. Néanmoins, on souhaite vérifier que l'utilisation d'un environnement industriel faciliterait l'intégration des outils et validerait aussi notre démarche. Ainsi, nous présentons dans la section suivante, l'intégration de notre flot dans l'outil Magillem.

5.7. Intégration du flot de génération de code de configuration dans l'environnement Magillem

Dans cette section, nous décrivons les étapes d'intégration de notre flot de génération de code dans l'environnement industriel Magillem, ainsi que les apports de cette nouvelle contribution. Le travail décrit dans cette thèse est réalisé dans le cadre du projet ARFU-HOSPI (HOmogeneous SPecification for Platform Integration) [Hospi] en partenariat avec le CEA-LETI et MDS (Magillem Design Services). Ainsi le choix de l'outil Magillem a été imposé et nous avons eu accès aux différentes fonctionnalités de l'outil.

5.7.1. Présentation de l'outil Magillem

L'environnement Magillem est un IDE qui fournit des services centrés sur le standard IP-XACT. Il est proposé par l'entreprise MDS (Magillem Design Services) [Magillem] et il est composé de plusieurs modules (Figure. 5.15) :

(1) MIP (Magillem IP-XACT Packager) : cet outil permet la représentation en IP-XACT des composants matériels, pour les niveaux RTL ou TLM. Il intègre des éléments d'import automatique à partir de modèles RTL ou de fichiers textes de type « .csv ». Il propose également une interface graphique permettant d'entrer manuellement les informations de description d'un bloc matériel en IP-XACT. Les fonctionnalités de ce module sont également accessibles en ligne de commande ou via l'interface TGI définie dans IP-XACT.

(2) MPA (Magillem Platform Assembly) : cet outil est utilisé pour décrire en IP-XACT les caractéristiques de la plateforme aux niveaux RTL ou TLM (instanciation et configuration des composants matériels, connexions de type ad-hoc ou de type interface, gestion des configurations). Il propose une interface schématique et permet notamment la génération des architectures dans différents langages (vhdl, verilog, systemC). Les fonctionnalités de ce module sont également accessibles en ligne de commande ou via l'interface TGI définie dans IP-XACT.

(3) MGS (Magillem Generator Studio) : ce module apporte toutes les fonctionnalités nécessaires au développement et au débogage des moteurs utilisant l'interface TGI de IP-XACT. Cela permet de programmer les différentes étapes qui interviennent dans un flot de conception et de vérification.

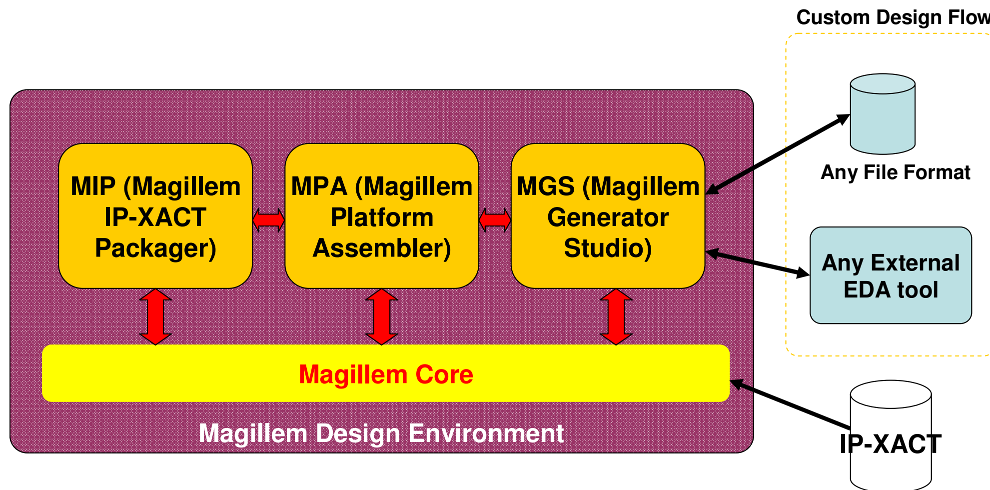


Figure 5.15. Environnement Magillem [Magillem]

Nous présentons dans la suite de ce chapitre l'approche que nous avons adoptée pour exploiter ces différents outils afin d'intégrer notre flot de génération de code de configuration dans l'environnement Magillem.

5.7.2. Construction d'une bibliothèque de composants matériels et assemblage des architectures

La phase de construction d'une bibliothèque de composants pour les architectures à base d'IPs est réalisée dans le module MIP (Magillem IP-XACT Packager). Ce dernier permet de créer les fichiers IP-XACT de description des composants matériels en utilisant les extensions que nous avons proposées (section 5.4.1). Ainsi, un même composant peut être décrit de différentes manières : ARDL et RTL par exemple. Chacune de ces représentations est considérée comme une vue du composant.

Le module MPA (Magillem Platform Assembly) permet l'assemblage de la plateforme en décrivant le modèle de l'architecture dans un élément IP-XACT « design » (instance des IPs, paramétrage des instances et interconnexions des interfaces de communication). Pour une instance d'un composant, disposant de plusieurs descriptions (ARDL, RTL, etc.), une vue peut être sélectionnée.

L'assemblage peut se faire manuellement, via l'interface schématique, ou automatiquement en exécutant un générateur basé sur l'API TGI de IP-XACT. Nous rappelons que la TGI est l'interface logicielle de type API qui permet l'interaction d'un programme ou d'un script avec les données IP-XACT. L'outil Magillem implante cette API.

5.7.3. Environnement de développement de la partie « Front-End » du flot de génération de code de configuration

Pour permettre le déploiement d'une application sur une plateforme modélisée en ARDL, nous avons développé un environnement en tant que « plugin » Eclipse et nous l'avons intégré dans l'outil Magillem.

Ce nouvel environnement, qui peut être qualifié d'environnement d'aide au partitionnement, utilise les fonctionnalités de Magillem en tâches de fond. La figure 5.16 présente ses différentes parties. Il fournit des fonctionnalités de gestion de projets qui rassemblent les représentations APDL des applications, ARDL des plateformes, le contrat de partitionnement et le modèle de partitionnement proprement dit.

Nous avons mis en place dans cet outil des assistants pour la création des modèles d'application et de partitionnement. Un éditeur graphique existe également pour décrire les contraintes associées au contrat de partitionnement (les contraintes spécifiques et les contraintes générales).

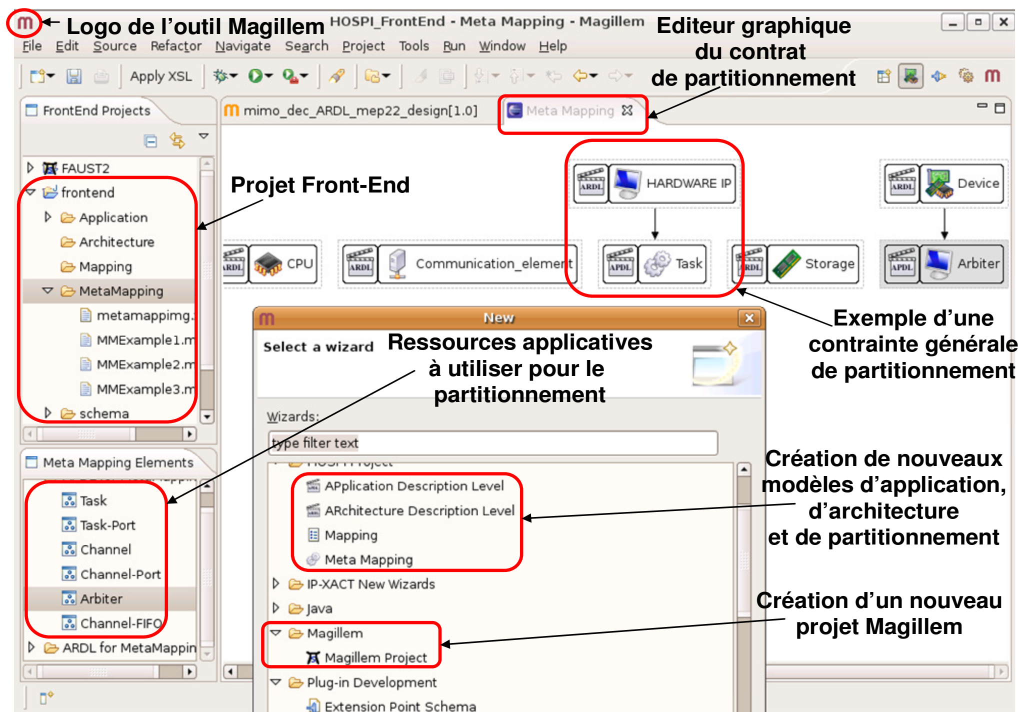


Figure 5.16. Interface graphique de l'environnement de modélisation de la partie "Front-End" du flot de generation de code de configuration

5.7.4. Développement des générateurs dans l'environnement Magillem

La création et l'exécution des composants « generator » pour les modèles intermédiaires « GIM » et « SIM » ainsi que pour le code de configuration et de simulation sont réalisées dans le module MGS (Magillem Generator Studio).

Une fois que les phases précédentes ont été effectuées (construction de la bibliothèque de composants, assemblage des instances des composants pour définir la plateforme), des composants « generator » sont exécutés pour la mise en œuvre du flot de génération du code

de configuration. L'outil Magillem fournit un éditeur graphique pour la création d'éléments « generator », ce qui facilite et accélère le processus de génération de code. Cet éditeur permet la définition et l'initialisation des paramètres du générateur y compris les exécutables des outils à lancer.

Le module MGS fournit également un environnement pour le développement des outils de génération qui sont référencés et appelés par les « generator ». Il consiste en une perspective « Eclipse » nommée « TGI Workshop ». Elle permet de gérer l'ensemble des générateurs et d'utiliser l'API TGI développée par Magillem, et qui s'appelle « TGIMDS » (Figure. 5.17.a).

Nous utilisons une autre API, que nous avons spécifiée, développée puis intégrée dans le module MGS, pour l'extraction des informations des modèles de représentation de l'application et du partitionnement (Figure. 5.17.b), en plus de l'API TGI (TGIMDS). Les outils de génération de code des modèles intermédiaires et du code de configuration peuvent être développés dans l'outil MGS.

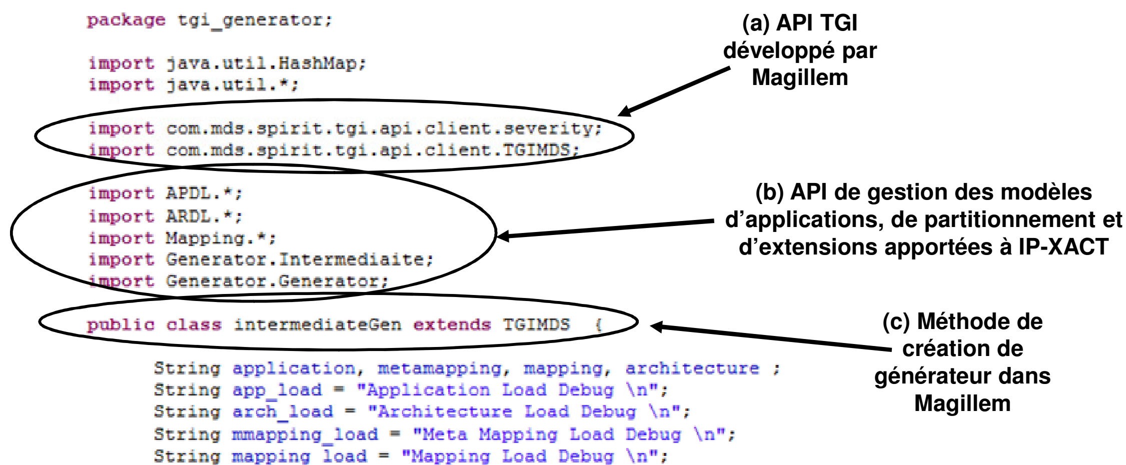


Figure 5.17. Développement des générateurs dans Magillem

Nous avons réussi à intégrer notre flot de génération de code à l'outil Magillem. Nous pouvons ainsi générer du code de configuration pour des plateformes à base d'IPs à l'aide de l'interface graphique de l'environnement que nous avons mis en place. Le passage de la représentation de haut niveau vers le code final est réalisé en faisant appel aux fonctionnalités de l'outil Magillem.

5.8. Conclusion

Dans ce chapitre, nous avons spécifié et implanté le flot de génération de code de configuration en se basant sur le standard IP-XACT.

Nous avons proposé plusieurs extensions au standard dans le but de réaliser les objectifs suivants : la description des composants matériels IPs, la description des chemins de communication physiques et l'automatisation du processus génération de code de configuration.

Nous avons montré également que nous pouvions intégrer ce flot dans un environnement industriel, et plus particulièrement Magillem pour exploiter ses fonctionnalités de définition, d'assemblage et de réutilisation de composants. Nous avons également décrit l'intégration de notre approche d'automatisation du flot de génération de code de configuration dans le module « MGS » de Magillem.

Nous allons présenter, au chapitre suivant, un exemple d'application de notre flot pour déployer une application de télécommunication sur la plateforme Magali.

Chapitre 6. Génération de code de configuration pour le déploiement d'une application de télécommunication sur la plateforme Magali

6.1. Modélisation de la plateforme Magali 87
6.1.1. Rappels des concepts de base de la plateforme Magali 87
6.1.2. Modélisation de la plateforme Magali dans l'environnement Magillem 88
6.2. Présentation de l'application de télécommunication 3GPP-LTE 95
6.2.1. Description de l'application 3GPP-LTE pour la génération de code de configuration 96
6.3. Partitionnement de l'application 3GPP-LTE sur la plateforme Magali 98
6.3.1. Partitionnement de la partie de calcul de l'application 98
6.3.2. Partitionnement de la partie de communication de l'application 100
6.4. Génération des modèles intermédiaires 101
6.4.1. Génération du modèle « GIM » 101
6.4.2. Génération du modèle « SIM » 102
6.5. Génération du code de configuration pour le déploiement de l'application 3GPP-LTE sur la plateforme Magali 107
6.5.1. Construction des catalogues des configurations de communication et de calcul 108
6.5.2. Construction des configurations de contexte 109
6.5.3. Code de configuration non généré 110
6.6. Simulation des fichiers de configuration 111
6.7. Conclusion 112

Nous présentons dans ce chapitre une application du flot de génération de code de configuration que nous avons proposé dans la perspective de déployer des applications de télécommunication sur la plateforme Magali. Nous rappelons dans un premier temps quelques concepts concernant cette plateforme. Nous expliquons, dans un deuxième temps, comment nous l'avons modélisée avec la représentation ARDL. Puis, nous présentons l'application de télécommunication 3GPP-LTE. Nous la décrivons avec la représentation APDL. Nous décrivons par la suite les modèles de partitionnement ainsi que les mécanismes d'automatisation du flot de génération de code de configuration. Nous montrons comment nous avons mis en œuvre notre approche en utilisant l'environnement Magillem. Enfin, nous décrivons la simulation d'un modèle SystemC de la plateforme utilisant les fichiers de configuration et nous présentons quelques premiers résultats.

6.1. Modélisation de la plateforme Magali

6.1.1. Rappels des concepts de base de la plateforme Magali

Le système sur puce Magali, utilisé comme cas d'étude, est un circuit dédié à la réalisation des couches physique et MAC (Media Access Control) d'applications de télécommunication haut débit, de type 3G et suivantes (3GPP-LTE, WiMax, etc.). La plateforme contient un ensemble d'IPs fournissant diverses fonctionnalités spécifiques. Chaque IP est relié à un routeur par l'intermédiaire d'un bloc CCC (*Communication and Configuration Controller*) qui étend les fonctionnalités d'une interface réseau classique à travers son gestionnaire des communications et d'exécution de tâches.

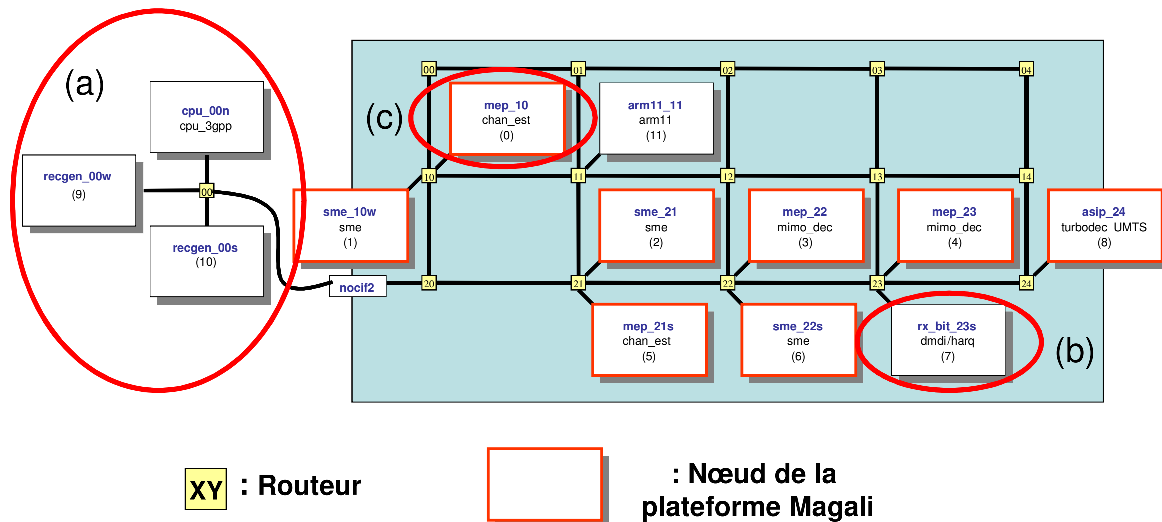


Figure 6.1. Sous-ensemble de Magali utilisé pour simuler l'exécution d'applications

Plusieurs configurations de calcul et de communication sont définies pour décrire le comportement global de l'application. Il s'agit de modéliser et de séquencer les différents échanges de données entre les unités fonctionnelles. Ces configurations sont exécutées par les interfaces réseau configurables. Plus de détails sur le circuit ainsi que ses mécanismes de fonctionnement et de configuration sont dans le chapitre 3 de l'état de l'art.

Dans cette étude de cas, nous nous sommes limités à une sous-partie représentative du circuit Magali (Figure.6.1). Les trois unités sur la gauche de la plateforme (Figure.6.1.a) permettent de simuler l'application : deux générateurs/récepteurs de données constituent d'une part la source des données à traiter (« recgen_00w ») et d'autre part, la destination des données issues de l'application. Le processeur « cpu_3gpp » contrôle le fonctionnement de l'ensemble de la plateforme et de l'application (« recgen_00s »). Il s'agit du processeur d'orchestration.

L'application à déployer est distribuée sur les différents nœuds de la plateforme grâce aux séquenceurs locaux des blocs CCC. Ces derniers permettent la minimisation des interventions du processeur central d'orchestration. Le contrôle distribué de l'application améliore grandement les performances. L'inconvénient qui accompagne ce gain réside dans la complexité de la programmation de l'application. Elle nécessite la production de configurations cohérentes entre elles et distribuées sur plusieurs entités (mémoires, registres) à différents endroits du réseau sur puce.

La plateforme Magali est utilisable et simulable de deux manières:

(1) Simulation par fichiers de configuration : les configurations des différentes unités sont stockées dans des fichiers sous un format textuel et lisible. Des scripts décrivant le déroulement de l'application sont exécutés par le processeur central d'orchestration. Ils gèrent les contrôleurs de communication et de configuration (les blocs CCC). Ce cas est principalement utile pour la mise au point du circuit et pour le test unitaire des IPs.

(2) Simulation par « compilation » : Cette seconde stratégie de simulation fait appel à une compilation d'une description des flux de données applicatifs et des configurations des unités. Ces dernières sont rangées dans des tables au format binaire, dans les composants SME de la plateforme, qui sont des composants de mémorisation programmables. Nous les détaillons dans la suite de ce chapitre. Un exécutable ARM est également développé. Il se charge d'effectuer les initialisations nécessaires au lancement de l'application et correspond au comportement du processeur d'orchestration.

L'outil « ComC », présenté dans le chapitre 3, permet la génération des configurations binaires correspondant à la deuxième méthode de simulation. Nous utilisons notre flot de génération de code de configuration pour générer les configurations lisibles et utilisables par la première méthode. Le code est ainsi constitué de plusieurs fichiers de configuration répartis sur plusieurs composants matériels de la plateforme. Ces configurations ainsi que leurs répartitions sur les composants de la plateforme ont été décrits dans le chapitre 3.

6.1.2. Modélisation de la plateforme Magali dans l'environnement Magillem

Nous modélisons la plateforme Magali en se basant sur le standard IP-XACT et sur les extensions que nous lui avons rajoutées. Pour ce faire, nous sommes passés par deux phases : la construction d'une bibliothèque de composants et l'assemblage de la plateforme.

6.1.2.1. Construction de la bibliothèque de composants

Nous utilisons l'outil MIP (Magillem IP-XACT Packager) de l'environnement Magillem afin de construire une bibliothèque de composants de la plateforme Magali.

Au départ, nous disposons d'une description IP-XACT de la plateforme Magali au niveau RTL qui nous a été fournie par le CEA-LETI et MDS. Cette dernière permet la génération du code SystemC de simulation de la plateforme. Cependant, cette représentation ne contient pas tous les détails que nous souhaitons modéliser dans une plateforme telle que Magali. Elle ne permet pas de représenter, par exemple, des informations concernant le trafic des données applicatives entre les tâches et les IPs. Elle ne décrit pas non plus les chemins de communication et par conséquence ne peut pas servir pour la génération des configurations nécessaires à plateforme Magali. Ces fichiers de configuration devaient être écrits de façon manuelle (environ 80 pour l'application de télécommunication que nous présentons dans la suite de chapitre), et étaient source d'erreurs difficile à déboguer.

Notre approche repose sur un modèle de la plateforme comme entrée. Pour cette raison, nous proposons une représentation ARDL de la plateforme Magali, en plus de celle en RTL, pour générer le code de configuration. Ce dernier peut être exécuté et validé par un modèle de simulation écrit en SystemC.

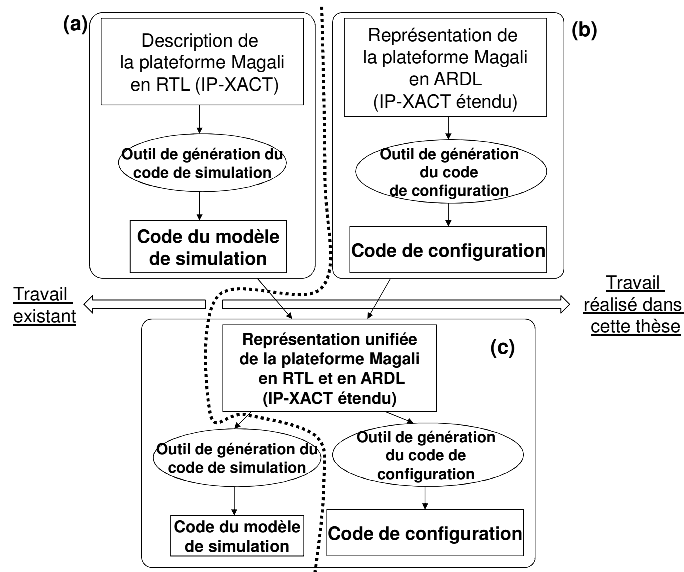


Figure 6.2. Représentation RTL et ARDL unifiée de la plateforme Magali pour la génération du code du modèle de simulation et du code de configuration

Ainsi la plateforme Magali dispose de deux descriptions : Une en RTL qui est déjà disponible (Figure. 6.2.a) et une en ARDL (ARchitecture Description Level) que nous développons (Figure. 6.2.b). Nous fournissons, à travers le mécanisme de vue d'IP-XACT, une description unifiée (Figure. 6.2.c) de la plateforme qui englobe la vue RTL et la représentation ARDL. Cela permet de générer en même temps le modèle de simulation SystemC et le code de configuration.

a) Représentation ARDL des composants de l'architecture matérielle de Magali

La représentation ARDL utilise IP-XACT et les «vendorExtensions » que nous avons rajoutés au standard. Ainsi un composant matériel est défini avec ses interfaces de communication « BusInterfaces » utilisant les ports TLM d'IP-XACT, ses fichiers (« FileSets »), et une extension ARDL. Elle définit la famille du composant (composant de type « HWIP » dans l'exemple illustré dans la figure 6.3.b), ses paramètres et la liste des fonctionnalités qu'il fournit (balise « HALAPI »).

On crée les composants de type «NI» pour les interfaces réseau, les composants « HWIP » pour l'abstraction des composants IPs de calcul et deux composants périphériques de contrôle « ICC » et « OCC » (Figure. 6.3.A). Ils servent à abstraire les mécanismes de gestion des communications dans la plateforme Magali.

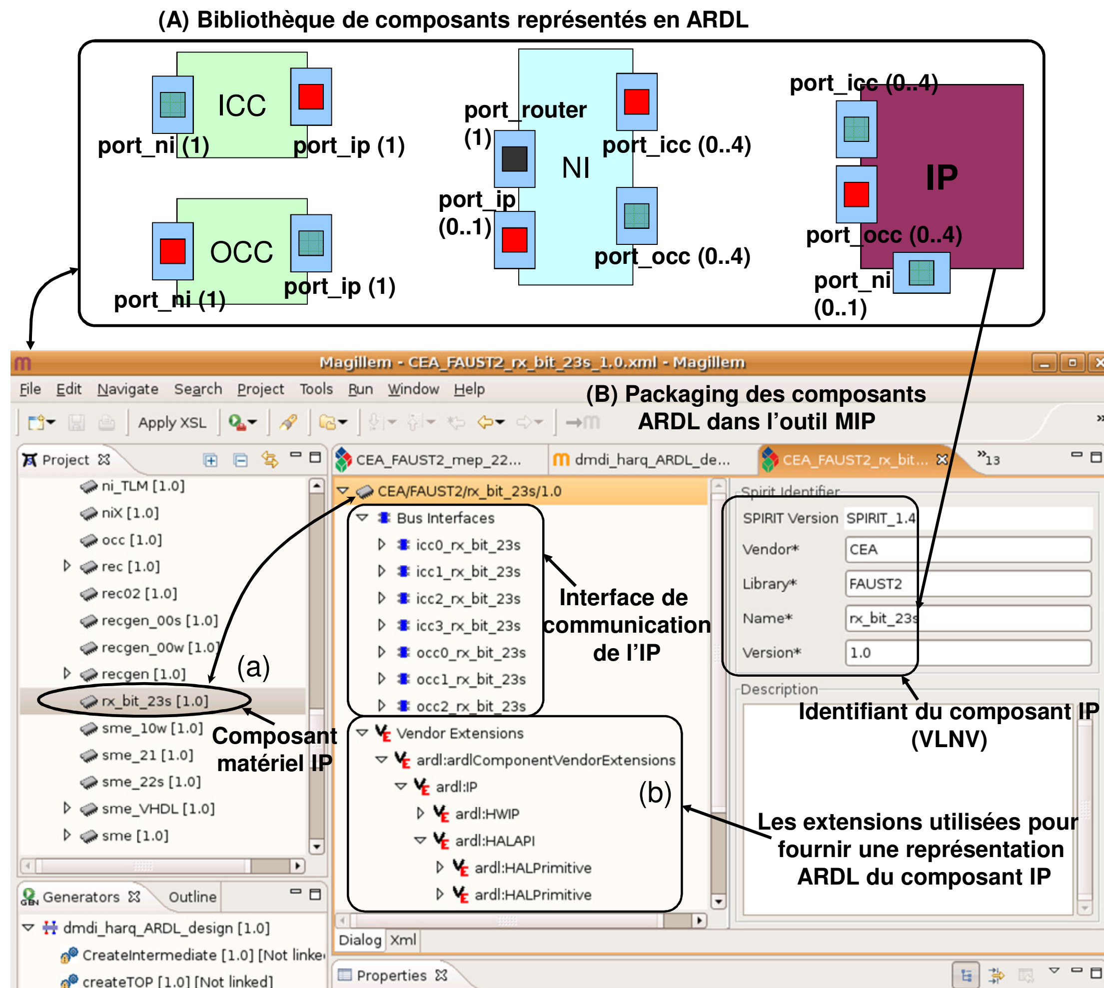


Figure 6.3. Définition de la bibliothèque de composants ARDL pour l'abstraction de la plateforme Magali

Un exemple est illustré dans la figure 6.3.a montrant la description d'un composant de type « HWIP » nommé « rx_bit_23 ». Il dispose de quatre ports d'entrée (icc0_rx_bit_23s..icc3_rx_bit_23s) et trois de sortie (occ0_rx_bit_23s.. occ2_rx_bit_23s). Les

ports d'entrée sont interconnectés aux périphériques « ICC » et ceux de sortie sont interconnectés aux « OCC ». La totalité des ports est définie dans l'interface de communication « BusInterfaces ». On spécifie le type de ce composant ainsi que ses caractéristiques matérielles grâce à la balise « ardl:HWIP » (Figure. 6.3.b) appartenant au «VendorExtensions». Les fonctionnalités fournies par l'IP sont décrites dans la balise « ardl:HALAPI ». Cette dernière contient plusieurs sous balises « ardl:HALPrimitive » qui décrivent les fonctionnalités à travers une liste de paramètres (paragraphe 5.4.1). La figure 6.3.A illustre les différents types de composants matériels que nous définissons dans la bibliothèque en utilisant l'outil MIP (Magillem IP-XACT Packager).

b) Représentation ARDL d'un nœud interconnecté au réseau sur puce

Une fois que nous avons défini les composants de base de la plateforme Magali (interface réseau : « NI », périphériques de contrôle : « ICC » et « OCC » et les composants IP : « HWIP »), nous spécifions comment ils sont interconnectés les uns aux autres. Un nœud ARDL dans la plateforme Magali (Figure. 6.4) comporte une seule interface réseau (NI), des contrôleurs de communication de type « OCC » et « ICC » au nombre maximum de 4 pour chacun, et un unique composant IP.

L'interface réseau (NI) dispose d'un port d'interconnexion avec le routeur, de quatre ports d'entrée (port_occ0..3), de quatre ports de sorties (port_icc0.. 3) et d'un port « port_ip » pour une éventuelle interconnexion directe avec l'IP. Le bloc CCC est modélisé avec au maximum quatre composants périphérique « ICC » et quatre « OCC ». Un « ICC » a un port d'entrée « port_ni » et un de sortie « port_ip ». L'élément « OCC » a également un port d'entrée « port_ip » et un de sortie « port_ni ». Le composant « HWIP », quant à lui, dispose au maximum de 4 ports d'entrée liés aux composants « ICC » et de 4 en sortie liés aux composants « OCC ».

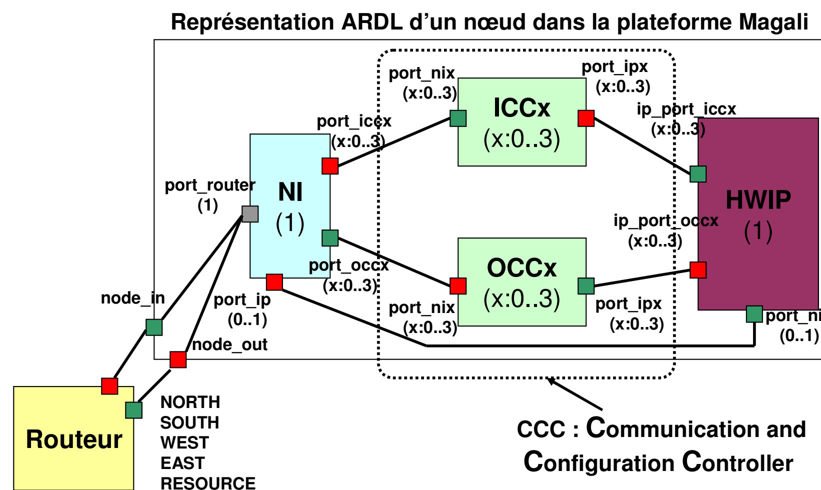


Figure 6.4. Méta modèle d'un nœud matériel abstrait en ARDL pour la plateforme Magali

La figure 6.5 illustre un exemple de description d'un nœud « mep_10 » (Figure. 6.5.a/6.5.b) interconnecté au routeur « 10 » de la plateforme Magali.

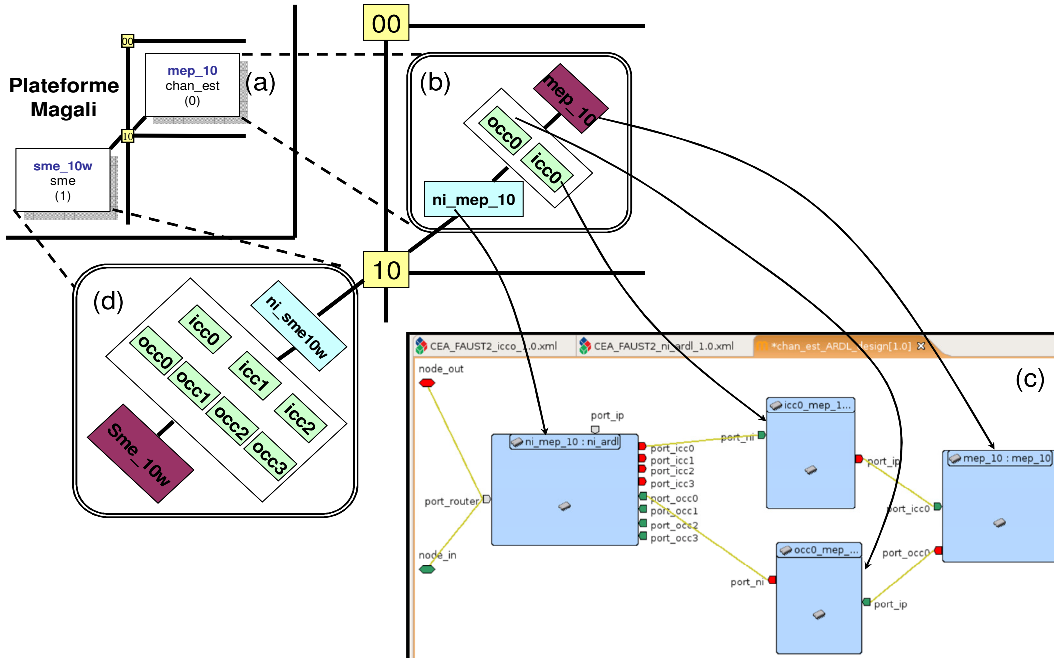


Figure 6.5. Représentation ARDL d'un exemple d'un nœud matériel avec l'outil Magillem

Le nœud schématisé en Figure. 6.5.b contient une interface réseau « ni_mep_10 », un périphérique de contrôle « OCC » nommé « occ0_mep10 » et un périphérique « ICC » nommé « icc0_mep10 ». Il dispose également d'un composant IP nommé « mep_10 ».

L'interface réseau « ni_mep10 » est interconnectée aux périphériques de contrôle « occ0_mep10 » et « icc0_mep10 » à travers les ports « port_occ0 » et « port_icc0 ». La figure 6.5 illustre également la description du nœud avec l'éditeur graphique de l'outil Magillem (Figure. 6.5.c).

c) Unification des représentations ARDL et RTL d'un nœud dans la plateforme Magali

Une fois les différents nœuds de la plateforme Magali décrits en ARDL, nous les associons au modèle initial décrit en RTL et fourni par le CEA-LETI et MDS. Nous rappelons que le standard IP-XACT fournit la balise «spirit:view» dans le schéma « component.xsd » afin de regrouper plusieurs représentations d'un même composant.

La figure 6.6 schématise un exemple de nœud de la plateforme Magali décrit avec deux représentations (RTL, ARDL). La partie «Model» (Figure. 6.6.a) d'un «component» IP-XACT décrit les différentes vues (vue RTL et vue ARDL) disponibles.

Ainsi, pour assembler la plateforme Magali, son « design » importe les composants décrivant les nœuds ainsi que les routeurs du NoC. Ces derniers sont décrits uniquement en RTL. La représentation ARDL des nœuds est basée sur les composants définis dans (6.1.2.1.a et 6.1.2.1.b). La description en RTL sert pour la génération du modèle de simulation et nous utilisons ARDL pour aider à la génération du code de configuration. La figure 6.6 schématise le nœud «dmdi-harq» illustré dans la figure 6.1.b de la plateforme Magali.

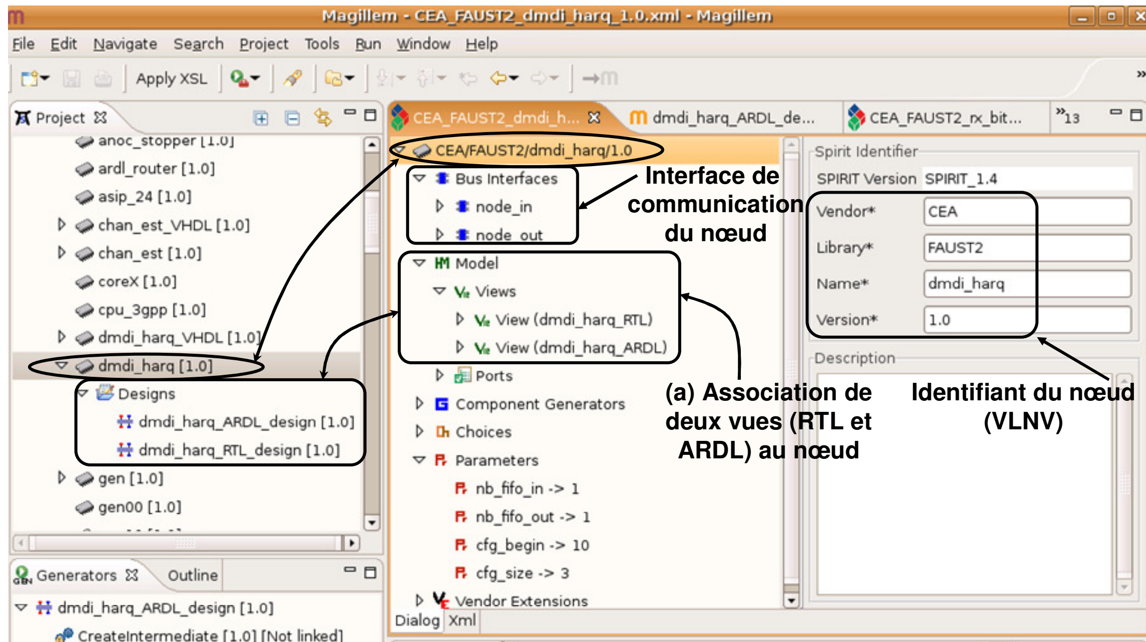


Figure 6.6. Représentation d'un exemple de nœud matériel avec l'outil Magillem

Nous avons réussi à ce stade à créer une bibliothèque de composants, qui englobe deux vues différentes (ARDL et RTL) des ressources de la plateforme Magali (IP, routeur, interface réseau, périphérique de contrôle). Chacune est utilisée pour la génération d'un type de code (code SystemC de simulation et code de configuration). Nous devons maintenant assembler la plateforme à partir de cette bibliothèque.

6.1.2.2. Assemblage de la plateforme Magali

La phase d'assemblage est réalisée dans le module MPA (Magillem Platform Assembly), qui permet de décrire en IP-XACT les instances des composants matériels ainsi que leurs paramétrages à travers l'élément « designConfiguration ». L'assemblage peut se faire manuellement, via l'interface schématique du MPA, ou automatiquement en exécutant un générateur IP-XACT.

a) Assemblage manuel de la plateforme Magali

L'assemblage manuel passe par la création d'un nouveau modèle « design » dans l'environnement Magillem. Il peut être écrit par un éditeur IP-XACT ou un éditeur graphique de l'environnement Magillem. Le premier est basé sur l'éditeur XML de l'environnement « Eclispe » et permet d'instancier et d'interconnecter les composants de l'architecture.

L'éditeur graphique facilite l'instanciation des composants matériels à travers des « Drag and Drop » depuis la bibliothèque de composants vers le « design ». Les composants sont représentés par des boites auxquels les ports sont attachés (Figure. 6.5.c). Nous avons opté pour une solution d'assemblage automatique pour la plateforme Magali, vu le grand nombre de composants à instancier et d'interconnexions à définir.

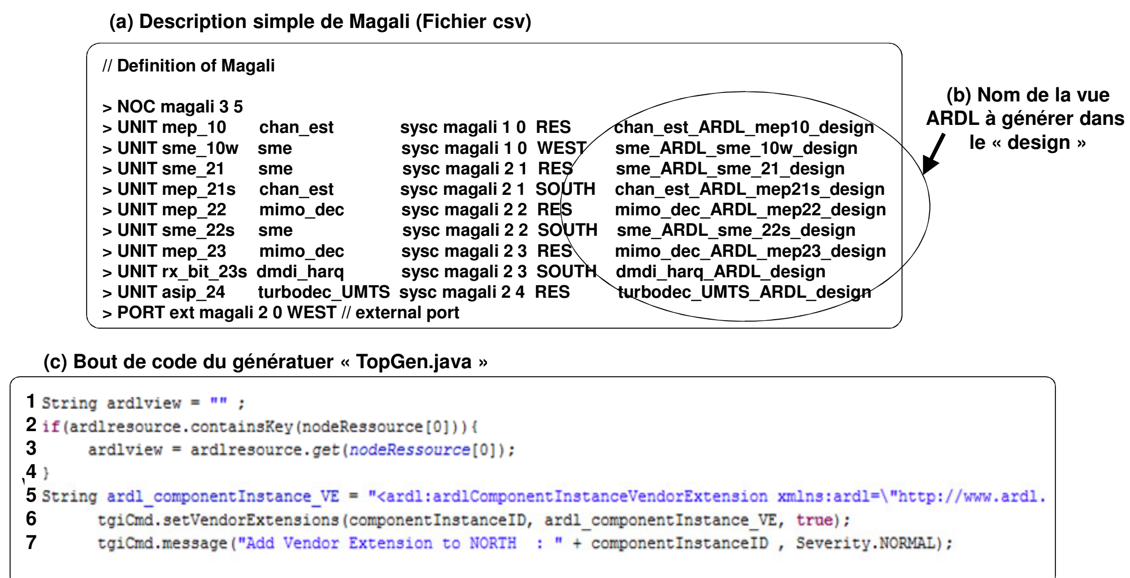
b) Assemblage automatique de la plateforme Magali

L'assemblage automatique des composants IP-XACT pour décrire la plateforme Magali est accompli à partir d'une description textuelle abstraite de la plateforme. Il est réalisé en utilisant un composant IP-XACT de type « generator ». Ce dernier prend en entrée un fichier texte (« csv ») qui contient une description simple de la plateforme et produit en sortie un « design » de la plateforme Magali, ainsi que son code SystemC de simulation. Cette solution a été initialement proposée par MDS pour générer automatiquement un « design » de la plateforme Magali au niveau RTL. Le même générateur intègre un outil développé par le « CEA-LETI » pour la génération du modèle de simulation SystemC de la plateforme.

Nous avons réutilisé cette solution et nous avons apporté des modifications à la description simple de la plateforme en « csv » ainsi qu'au générateur du « design ». Le fichier « csv » est un tableur qui contient des données, sur chaque ligne, séparées par un caractère de séparation. Celui développé par « MDS » commence chaque ligne par un mot clé (NOC, UNIT, PORT) (Figure. 6.7.a). Le mot clé « NOC » est suivi par des données relatives au nom du réseau ainsi que sa taille. Le mot clé « UNIT » concerne les nœuds interconnectés au réseau. Il est suivi par le nom du nœud, sa position sur le réseau, sa direction par rapport au routeur, etc. Le mot clé PORT concerne les interconnexions internes et externes du réseau sur puce.

Nous avons ajouté un champ en fin de chaque ligne contenant le mot clé « UNIT » qui précise la vue ARDL du nœud (Figure. 6.7.b) à instancier. Le « generator » prend en paramètre le fichier « csv » et le nom du « design » à générer. Nous avons modifié le code du générateur du « design » pour faire apparaître les chemins de communication physiques et les emplacements des instances des nœuds.

Un exemple du code du générateur « TopGen.java » est illustré dans la figure 6.7.c. On fait appel à la primitive « setVendorExtensions » de la TGI (ligne 6) afin d'ajouter les informations dédiées à ARDL dans le « design ».




```
// Definition of Magali

> NOC magali 3 5
> UNIT mep_10      chan_est       sysc magali 1 0  RES     chan_est_ARDL_mep10_design
> UNIT sme_10w     sme            sysc magali 1 0  WEST    sme_ARDL_sme_10w_design
> UNIT sme_21      sme            sysc magali 2 1  RES     sme_ARDL_sme_21_design
> UNIT mep_21s     chan_est       sysc magali 2 1  SOUTH   chan_est_ARDL_mep21s_design
> UNIT mep_22      mimo_dec       sysc magali 2 2  RES     mimo_dec_ARDL_mep22_design
> UNIT sme_22s     sme            sysc magali 2 2  SOUTH   sme_ARDL_sme_22s_design
> UNIT mep_23      mimo_dec       sysc magali 2 3  RES     mimo_dec_ARDL_mep23_design
> UNIT rx_bit_23s  dmdi_harq      sysc magali 2 3  SOUTH   dmdi_harq_ARDL_design
> UNIT asip_24     turbodec_UMTS  sysc magali 2 4  RES     turbodec_UMTS_ARDL_design
> PORT ext magali 2 0 WEST // external port
```

```
1 String ardlview = "" ;
2 if(ardlresource.containsKey(nodeRessource[0])){
3     ardlview = ardlresource.get(nodeRessource[0]);
4 }
5 String ardl_componentInstance_VE = "<ardl:ardlComponentInstanceVendorExtension xmlns:ardl=\"http://www.ardl.
6     tgiCmd.setVendorExtensions(componentInstanceID, ardl_componentInstance_VE, true);
7     tgiCmd.message("Add Vendor Extension to NORTH  : " + componentInstanceID , Severity.NORMAL);
```

Figure 6.7. Bout de code du générateur automatique d'un « design » de la plateforme MAGALI

La figure 6.8 illustre un bout du code SystemC du modèle de simulation de Magali que nous générons après l'exécution du « generator » « GenTop » (celui qui exécute « TopGen.java »). Ce dernier produit un module principal (Figure. 6.8.a) dans lequel il instancie les routeurs (Figure. 6.8.b) et les nœuds (Figure. 6.8.c). Des composants appelés «stopper» sont également instanciés et modélisent les directions non utilisées par les routeurs.

(a) Module principal du modèle de simulation de la plateforme Magali

(b) Déclaration des routeurs dans la plateforme

(c) Déclaration des ressources

```
void top_magaliARDL::anoc_create_top() {
  // read appli.name and noccfg.name files (res_utils)
  set_path_appli();
  set_path_noccfg();
  // Node creation
  magali_node_0_0 = new anoc_node("anoc_node_0_0", MAGALI_NODE_WAIT);
  magali_node_0_1 = new anoc_node("anoc_node_0_1", MAGALI_NODE_WAIT);
  magali_node_0_2 = new anoc_node("anoc_node_0_2", MAGALI_NODE_WAIT);
  magali_node_0_3 = new anoc_node("anoc_node_0_3", MAGALI_NODE_WAIT);
  magali_node_0_4 = new anoc_node("anoc_node_0_4", MAGALI_NODE_WAIT);
  magali_node_1_0 = new anoc_node("anoc_node_1_0", MAGALI_NODE_WAIT);
  magali_node_1_1 = new anoc_node("anoc_node_1_1", MAGALI_NODE_WAIT);
  magali_node_1_2 = new anoc_node("anoc_node_1_2", MAGALI_NODE_WAIT);
  magali_node_1_3 = new anoc_node("anoc_node_1_3", MAGALI_NODE_WAIT);
  magali_node_1_4 = new anoc_node("anoc_node_1_4", MAGALI_NODE_WAIT);
  magali_node_2_0 = new anoc_node("anoc_node_2_0", MAGALI_NODE_WAIT);
  magali_node_2_1 = new anoc_node("anoc_node_2_1", MAGALI_NODE_WAIT);
  magali_node_2_2 = new anoc_node("anoc_node_2_2", MAGALI_NODE_WAIT);
  magali_node_2_3 = new anoc_node("anoc_node_2_3", MAGALI_NODE_WAIT);
  magali_node_2_4 = new anoc_node("anoc_node_2_4", MAGALI_NODE_WAIT);

#ifdef TLM_DVFS_MODELISATION
// cut_off signals creation
// FIXME
#endif
  // Resource creation
  magali_res_1_0 = new magali_unit_1_0("mep_10", 0, MAGALI_RES_CLK_PERIOD, MAGALI_NODE_WAIT, FULL_SYSC);
  magali_res_1_0_WEST = new magali_unit_1_0_WEST("sme_10w", 1, MAGALI_RES_CLK_PERIOD, MAGALI_NODE_WAIT, F
  magali_res_2_1 = new magali_unit_2_1("sme_21", 2, MAGALI_RES_CLK_PERIOD, MAGALI_NODE_WAIT, FULL_SYSC);
   ...
```

Figure 6.8. Code généré du modèle de simulation de la plateforme Magali

A ce stade, nous avons réussi à élaborer une méthode automatique d'assemblage et de génération du modèle de la plateforme. Le code SystemC de simulation est également généré en même temps que le « design » de Magali. Cette méthode nous a apporté un gain en temps de conception, en comparaison avec la méthode manuelle.

6.2. Présentation de l'application de télécommunication 3GPP-LTE

L'application de type 3GPP-LTE que nous déployons sur la plateforme Magali est celle étudiée dans le cadre du projet ANR RNRT OPUS [Huy08]. Il s'agit d'une application de télécommunication qui permet le transfert de données à plus longue portée, avec une plus grande vitesse et un important débit. On se focalise sur la partie réception du lien descendant, du côté du récepteur mobile. Quatre antennes sont utilisées à l'émission, et deux à la réception (MIMO 4x2).

Nous avons récupéré cette application du « CEA-LETI » qui a réalisé un travail important pour la rendre sous la forme d'un ensemble de traitements indépendants, selon le type des données échangées (données pilotes pour l'estimation des canaux, données de signalisation et autres données utiles). Un nouveau travail a été fait par le « CEA-LETI » pour rendre les traitements applicatifs compatibles avec les capacités des ressources de Magali. Certains ont été séparés en plusieurs sous traitements. D'autres ont été rassemblés car ils ne se déroulent pas pendant les mêmes phases de décodage ou parce qu'ils sont exécutables par une

même ressource physique. L'application est orientée flots de données. Elle est caractérisée par de nombreux échanges de données entre les unités fonctionnelles et elle est illustrée dans la figure 6.9.

L'application 3GPP-LTE se déroule en trois phases : phase d'estimation de canal, décodage de la signalisation et décodage des données utiles. Pour chacune, certains échanges requièrent une connaissance préalable des tailles des flux véhiculés. Un ordonnancement et une synchronisation des échanges de données sont également indispensables pour exécuter l'application.

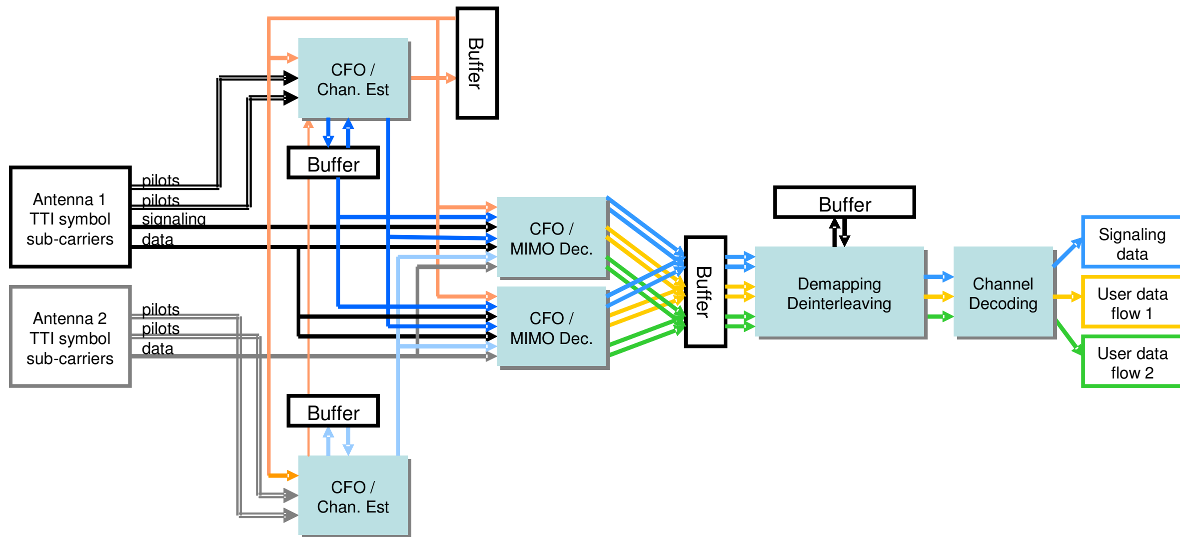


Figure 6.9. Description de l'application à déployer sur la plateforme Magali [Hospi]

Afin de déployer cette application sur la plateforme Magali, nous devons fournir une représentation des échanges des données ainsi que leurs ordonnancements et leurs synchronisations. Celles-ci servent à la configuration des contrôleurs de communication (CCC) associés à chaque IP de la plateforme. Le « CEA » nous a fourni une description de cette application à travers des figures de diagrammes de séquences qui ordonnancent les échanges des données entre les diverses tâches. Nous présentons dans la section suivante la représentation APDL de cette application.

6.2.1. Description de l'application 3GPP-LTE pour la génération de code de configuration

La représentation APDL de l'application 3GPP-LTE permet de décrire les tâches et les échanges de données entre elles. Nous rappelons qu'APDL modélise des canaux de communication multi-entrées multi-sorties, ce qui supporte la complexité des échanges rencontrée dans ce cas d'étude. L'application est décrite en entier en APDL dans la figure 6.10. Il s'agit d'un modèle graphique qui illustre la complexité du flot de données. Le format de ce modèle est en XML.

Un exemple d'utilisation d'un canal APDL multi-entrées, est illustré dans la figure 6.10.a. Il contient un arbitre d'entrée défini par une matrice. Cette dernière comporte un seul scénario d'entrée des données. Les deux premiers chiffres de l'unique colonne de la matrice

(1,1) référencent les deux ports d'entrée du canal. Le dernier chiffre de la colonne (24) représente le nombre de fois où le scénario d'écriture des données se répète. La description XML de ce canal de communication est illustrée dans la figure 6.10.b.

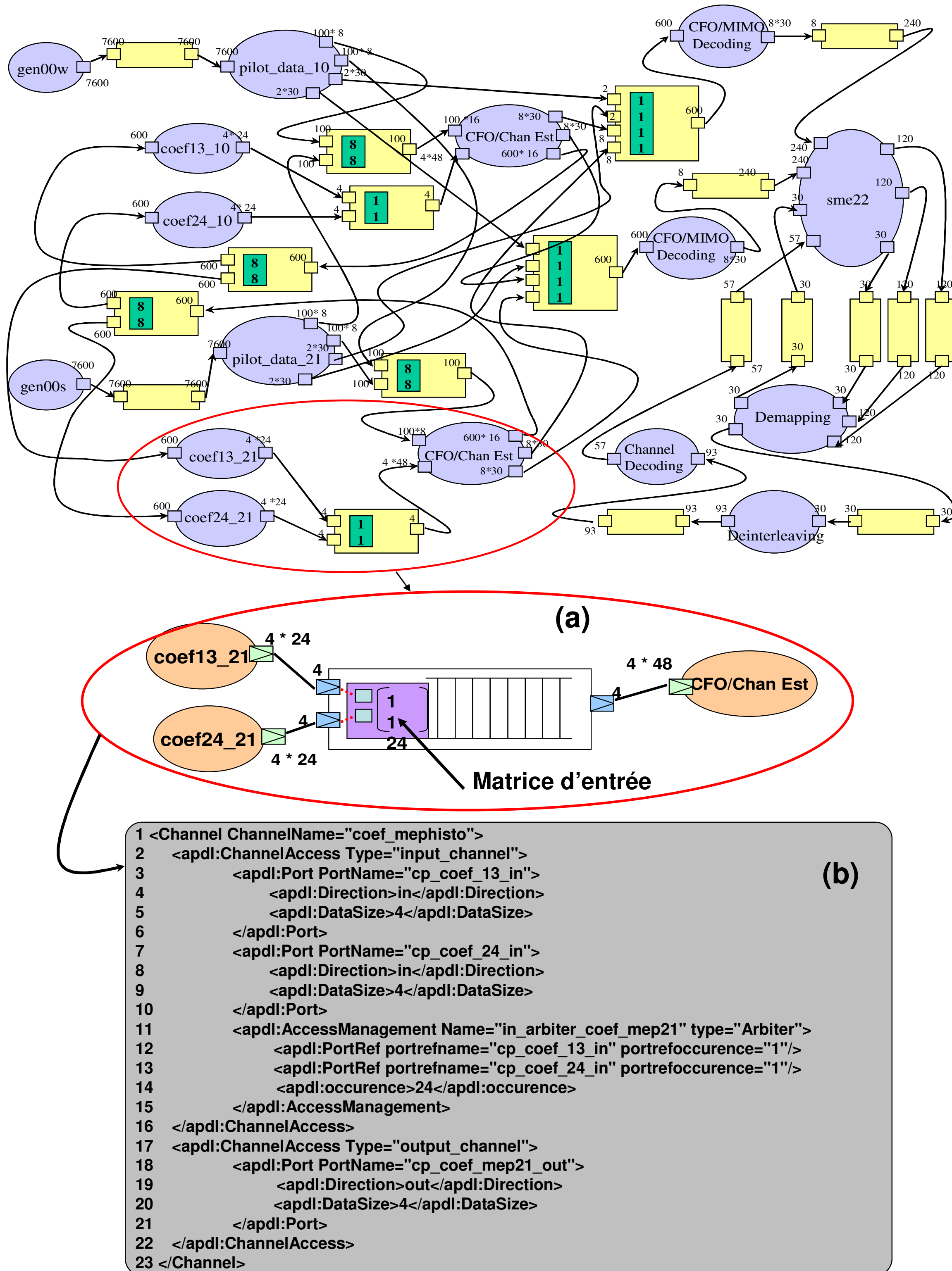


Figure 6.10. Représentation APDL de l'application 3GPP-LTE

Les ports d'entrée du canal sont décrits dans les lignes 3-10. L'arbitre d'entrée est défini dans les lignes 11-15 et le port de sortie dans les lignes 18-21. Le déroulement de ce scénario commence par l'écriture de 4 données dans la FIFO du canal depuis le premier port (connecté à la tâche « coef13_21 ») (Figure. 6.10.a). Par la suite, 4 autres données sont écrites dans la FIFO depuis le deuxième port (connecté à la tâche « coef24_21 »). Ce scénario est répété 24 fois. Les données transmises à la tâche « CFO/Chan Est » sont par lots de 4 données.

A ce stade, nous avons modélisé l'application 3GPP-LTE avec la représentation APDL, en utilisant des canaux de communication multi entrées, multi sorties avec l'arbitrage des données. Cette manière de décrire un flot de données n'est pas possible avec les langages qui respectent les modèles de calcul classiques comme KPN. La représentation de l'arbitrage des données est importante puisque ce dernier sera transformé en configurations de communication pour la plateforme Magali. La prochaine étape du flot est le partitionnement de l'application sur l'architecture.

6.3. Partitionnement de l'application 3GPP-LTE sur la plateforme Magali

Nous définissons le partitionnement de l'application 3GPP-LTE sur la plateforme Magali en deux principales étapes :

- Le contrat de partitionnement : Il s'agit de spécifier les contraintes de partitionnement liées à la plateforme Magali, qu'il faut respecter lors du placement et du déploiement de l'application 3GPP.
- L'étape de partitionnement : Il s'agit de l'étape classique de « mapping » où nous associons les ressources d'application modélisées en APDL aux ressources de la plateforme Magali modélisées en ARDL. Cette étape doit respecter les contraintes établies dans le contrat de partitionnement.

Ces deux étapes vont être appliquées aux deux parties de l'application, la partie de calcul et celle de communication.

6.3.1. Partitionnement de la partie de calcul de l'application

La partie de calcul de l'application consiste en l'ensemble des tâches applicatives qui doivent être exécutées sur les unités de calcul de la plateforme matérielle.

6.3.1.1. Contrat de partitionnement de la partie de calcul

Le calcul applicatif est modélisé en APDL par l'ensemble de tâches qui constituent l'application. Nous spécifions un ensemble de contraintes qui décrivent les règles de déploiement des fonctionnalités de l'application 3GPP-LTE sur les ressources de la plateforme Magali. Nous précisons, dans le contrat de partitionnement, que les tâches sont exécutées uniquement par des composants IP de la plateforme et non par l'unique processeur « ARM » d'orchestration. Nous souhaitons également restreindre l'exécution de certaines tâches à un nombre limité d'IPs.

Nous rappelons que les règles de partitionnement, telles que nous les avons spécifiées, sont de deux types : les règles générales qui établissent le placement des ressources applicatives sur le matériel, et les règles spécifiques liées aux instances des composants de la plateforme.

Nous établissons une contrainte générale (Figure. 6.11) autorisant le déploiement des tâches sur les IPs. Elle précise que les composants de type « HWIP » de la représentation ARDL peuvent exécuter des tâches APDL (lignes 3 et 4). L'absence de contraintes autorisant l'exécution des tâches par des processeurs, dans le contrat de partitionnement, signifie que ce type d'association n'est pas permis.

```
1 <GeneralMappingRules>
2         <MappingConstraint  name=« magali_resource_mapping »>
3                 <hardware_ResourceType>HWIP </hardware_ResourceType>
4                 <application_ResourceType>task_mapping</application_ResourceType>
5         </ MappingConstraint>
6         …
7 <GeneralMappingRules>
```

Figure 6.11. Contrainte générale autour du partitionnement des tâches APDL

Les contraintes spécifiques sont utilisées pour des instances de composants de l'application et/ou de la plateforme. La figure 6.12 établit une contrainte spécifique. Elle précise que la tâche « CFO/Chan Est » (ligne 3) ne peut s'exécuter que sur un ensemble bien défini de composants IPs (mep_10, mep_21s, mep_22, mep_23 : lignes 4..7).

```
1 <SpecificMappingRules>
2   <MappingConstraint  name=« CFO/Channel_mapping »>
3         <application_ResourceInstance>CFO/Chan Est </application_ResourceInstance>
4         <hardware_ResourceInstance>mep_10</hardware_ResourceInstance>
5         <hardware_ResourceInstance>mep_21s</hardware_ResourceInstance>
6         <hardware_ResourceInstance>mep_22</hardware_ResourceInstance>
7         <hardware_ResourceInstance>mep_23</hardware_ResourceInstance>
8   </ MappingConstraint>
9   …
10 <SpecificMappingRules>
```

Figure 6.12. Contrainte spécifique pour le mapping de la tâche « CFO/Chan Est »

6.3.1.2. Modèle de partitionnement de la partie de calcul

Le modèle de partitionnement associe les ressources applicatives aux ressources d'architecture en respectant le contrat précédemment établi.

Comme énoncé dans le contrat de partitionnement, les tâches ne peuvent être exécutées que sur certains IPs de la plateforme Magali. Les contraintes spécifiques guident la personne qui réalise le partitionnement à déployer les tâches sur les IPs adéquats. Dans la figure 6.13, la tâche « CFO/Chan Est » est associée au composant « mep_10 ». Ce déploiement respecte bien les contraintes générales (Figure. 6.11) et spécifiques (Figure. 6.12) du contrat de partitionnement. Un partitionnement analogue doit être réalisé pour toutes les tâches de l'application.

```
1 <TaskBinding TaskRef="CFO/Chan Est">
2    <ressource>
3            <HWIPRef>mep_10</HWIPRef>
4    </ressource>
5            <TaskPortBinding …
6 </TaskBinding>
```

Figure 6.13. Association de la tâche « CFO/Chan Est » au composant IP « mep_10 »

6.3.2. Partitionnement de la partie de communication de l'application

La partie de communication de l'application concerne les échanges de données entre les tâches applicatives et leur les arbitres et les ports de canal.

6.3.2.1. Contrat de partitionnement de la synchronisation à travers partie communication

Nous spécifions un ensemble de contraintes qui décrivent les règles de déploiement des ressources de communication applicatives décrites en APDL sur les ressources ARDL de la plateforme Magali.

Les informations sur le flux de données sont présents dans les ports de tâches et dans les canaux de communication logiciels (arbitres, ports de canaux). Nous rappelons que la plateforme Magali dispose de périphériques de contrôle de type « ICC » et « OCC » définissant les échanges de données entre les différents IPs du réseau.

Une contrainte générale (Figure. 6.14) est établie pour autoriser le déploiement des ports de tâches applicatives et les arbitres des canaux de communication sur les périphériques « ICC/OCC » de la plateforme Magali.

```
1   <MappingConstraint  name=« magali_CC_mapping »>
2           <hardware_ResourceType>CC</hardware_ResourceType>
3           <application_ResourceType>task_port_mapping</application_ResourceType>
4          <application_ResourceType>arbiter_mapping</application_ResourceType>
5   </ MappingConstraint>
```

Figure 6.14. Règle générale de partitionnement pour les composants ICC/OCC de la plateforme Magali

6.3.2.2. Modèle de partitionnement de la partie communication

Le modèle de partitionnement définit l'association des canaux applicatifs aux chemins de communication physiques. Ces derniers sont décrits par les IPs source et destinataire ainsi que les périphériques ICC et OCC qui assurent cette communication (IP_source → OCC → ICC → IP_destinataire). La figure 6.15 illustre le déploiement d'un canal applicatif de l'application 3GPP-LTE nommé « pilot_data_13 » (ligne 1). Il est associé à deux chemins de communication physiques :

« sme_10w_occ0_icc0_mep_10 » et « sme_21_occ0_icc0_mep_10 » (lignes 2 et 3).

L'application de la contrainte générale dans la Figure. 6.15, autorise le déploiement de l'arbitre du canal « pilot_data_13 » sur le composant « ICC0 » associé à l'IP « mep_10 ». Cet IP est référencé par les chemins de communications « sme_10w_occ0_icc0_mep_10 » et « sme_21_occ0_icc0_mep_10 ».

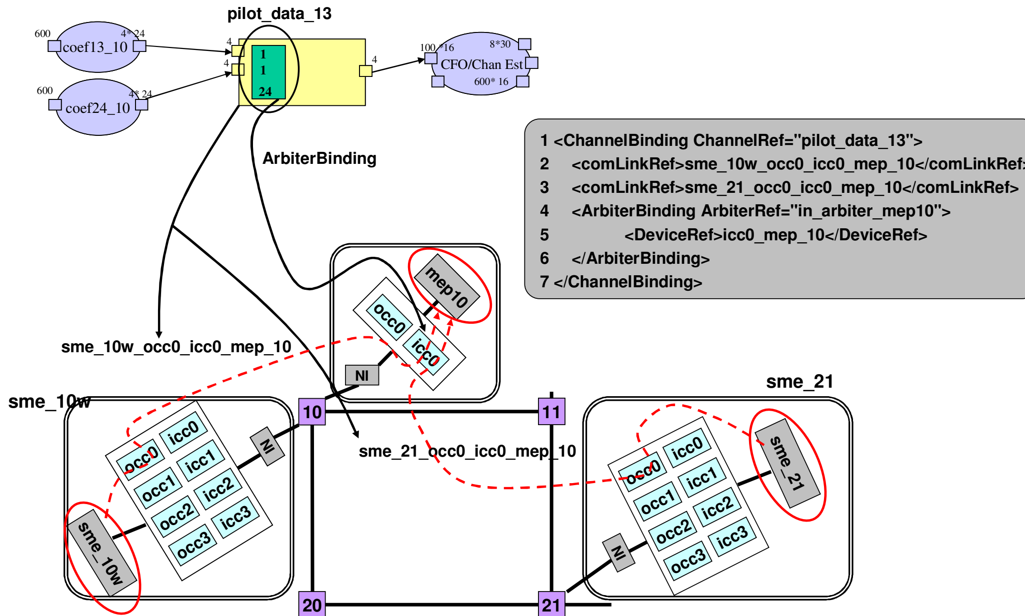


Figure 6.15. Partitionnement d'un canal APDL sur la plateforme Magali

Dans d'autres types de canaux de communication APDL ayant une seule entrée et une seule sortie, les ports d'entrée des canaux sont simplement associés aux périphériques ICC et les ports de sortie aux périphériques OCC.

A ce stade, nous avons validé et mis en œuvre nos modèles de partitionnement pour l'étude de cas de la plateforme Magali. Cette dernière a des caractéristiques particulières (programmation des interfaces réseau configurables) qui nécessitent un modèle de partitionnement très spécifique. Nous avons réussi à exprimer les contraintes de la plateforme dans le contrat de partitionnement. Cela nous a servi pour décrire le partitionnement d'une manière correcte, en prenant en considération les spécificités de Magali. La vérification de la conformité du modèle de partitionnement par rapport au contrat de partitionnement est réalisée par un outil que nous avons développé. Nous passons à l'étape suivante de notre flot, qui est la génération des modèles intermédiaires « GIM » et « SIM ».

6.4. Génération des modèles intermédiaires

6.4.1. Génération du modèle « GIM »

La génération du modèle intermédiaire GIM passe par l'exécution de l'élément IP-XACT « generator » intitulé « genGim » que nous avons présenté dans le paragraphe 5.6.1. Il est associé à la représentation ARDL de la plateforme Magali. Il prend comme paramètres d'entrée la représentation APDL de l'application 3GPP-LTE (paragraphe 6.2.1), le contrat de partitionnement et le modèle de partitionnement des ressources de calcul et de communication

(paragraphe 6.3). Le modèle « GIM » généré inclut les composants applicatifs APDL déployés sur les composants matériels de la plateforme Magali décrits en ARDL.

6.4.2. Génération du modèle « SIM »

6.4.2.1. L'élément « genSim » de type IP-XACT « generator »

Le « generator » IP-XACT « genSim », présenté dans le paragraphe 5.6.1, permet la génération du modèle SIM. Il prend comme paramètres d'entrée le modèle intermédiaire GIM et les informations spécifiques à la plateforme, qui sont modélisées dans un élément IP-XACT « designConfiguration » enrichi avec les extensions que nous lui avons rajoutées (voir paragraphe 5.4.1). Dans ces informations, nous associons à chaque composant matériel du GIM un composant simulable (SystemC). Les composants simulables écrits en SystemC (et fourni par le CEA-LETI) pour la plateforme Magali sont « chan_est », « sme », « mimo_dec », etc. Les IPs « mep_10 » et « mep_21s » sont référencés par le composant « chan_est ». La syntaxe des configurations est également présente dans les informations spécifiques. Il s'agit de définir la liste des paramètres à initialiser pour chaque type de configuration (ICC, OCC, IP, etc.).

Dans la suite, nous présentons un exemple particulier d'information spécifique à la plateforme Magali qui concerne les composants de mémorisation programmables SME. Nous commençons d'abord par présenter leurs caractéristiques.

Un SME (Smart Memory Engine) (Figure 6.16) est considéré comme un DMA programmable. Il dispose d'une mémoire divisée en 4 tampons (buffers) (Figure. 6.16.a) qui stockent les données. Il contient également une zone mémoire qui stocke des programmes écrits avec un langage pseudo-C, dont l'extension est « c_sme » (Figure. 6.16.b). Ces derniers traitent et réorganisent les données se trouvant dans les tampons.

La gestion des données dans les tampons se fait à travers des pointeurs (rd, wk, wr). Les composants ICC (Figure. 6.16.c) en entrée sont liés à des unités « PW : Process Writer ». Ces dernières assurent la transmission des données entre l'ICC et le tampon concerné du SME (buffer0, buffer1, buffer2, buffer3).

Une unité appelée « FWL » exécute le programme «.c_sme » suite à des requêtes provenant de l'unité « PR : Process Writer ». Cette dernière est liée aux canaux physiques de type « CH ». Il existe en tout quatre unités « PR ». Chacune est liée aux quatre canaux physiques du composant SME à travers quatre canaux logiques (Figure. 6.16.d). Les périphériques « OCC » (Figure. 6.16.e) liés aux composants SME sont différents des « OCC » classiques associés aux autres IPs. En effet, ils disposent de quatre sorties. Ainsi, nous avons besoin de connaître la direction de sortie depuis l'OCC (0, 1, 2, 3) pour programmer les communications du SME. L'association des canaux logiques aux canaux physiques apparaissent dans le fichier de configuration « core.cfg » du composant (Figure. 6.16.f). La structure et la grammaire des configurations « occ » (Figure.16.g) et « core » (Figure. 6.16.f) pour les composants de type « SME » est particulière par rapport aux autres types de composants.

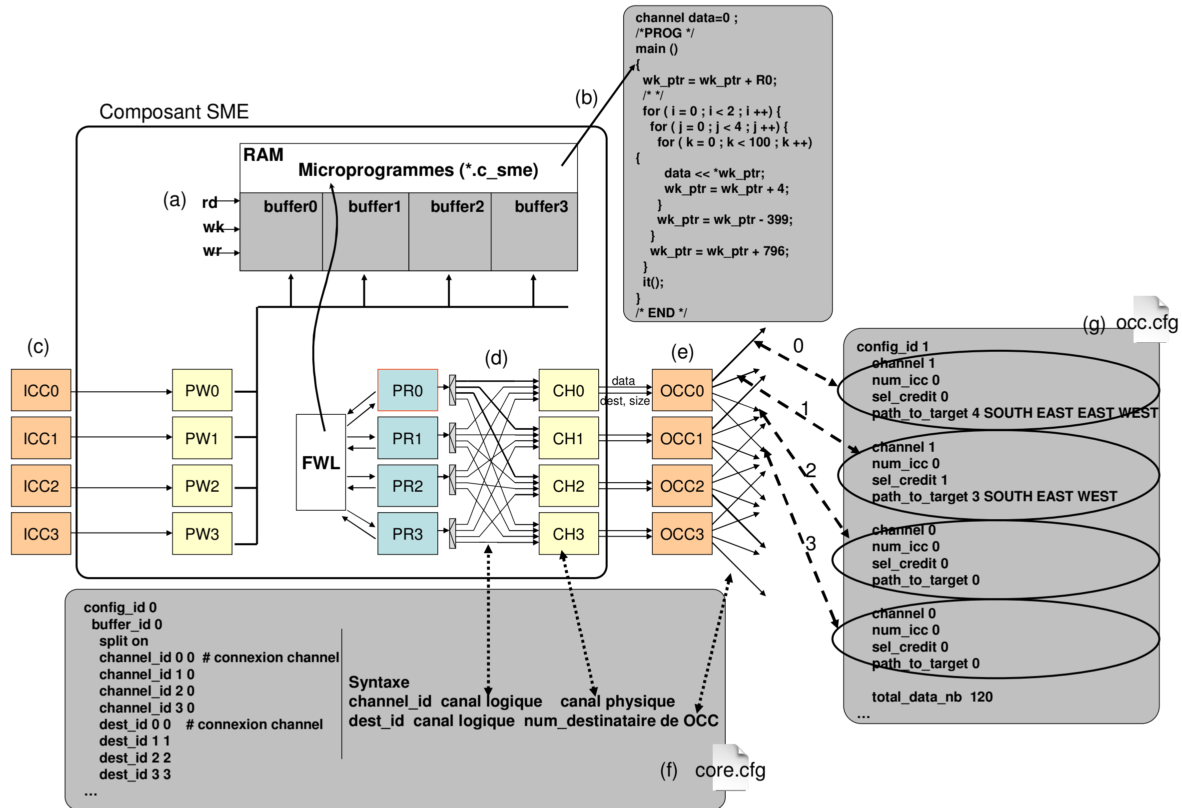


Figure 6.16. Structure d'un composant SME (Smart Memory Engine)

Les informations spécifiques aux SME (Figure. 6.17) définissent la mise en correspondance entre les variables de type « channel » dans les programmes «.c_sme » (Figure. 6.16.b) et les tampons sur lesquels ces programmes sont exécutés (Figure. 6.17 ligne 1). Elles précisent le chemin ou la direction à prendre à travers l'OCC (canal virtuel et une des quatre sorties de l' « OCC ») pour atteindre l'IP destinataire (Figure. 6.17 lignes 3,4 et 7). Elles ont une structure particulière et nous les avons décrits séparément du « designConfiguration ».

```
1 <buffer_id>0</buffer_id>
2 <microprogram name="read_data">
3           <virtual_channel name="data0" destination="mep_22">0</virtual_channel>
4           <virtual_channel name="data1" destination="mep_23">1</virtual_channel>
5 </microprogram>
6 <microprogram name="read_pilot">
7          <virtual_channel name="data" destination="mep_10">0</virtual_channel>
8 </microprogram>
```

Figure 6.17. Informations spécifiques à la plateforme Magali concernant les composants SME

Le générateur « genSim » est constitué de deux phases. Une qui traite les données provenant du modèle « GIM » et une qui traite le « GIM » avec les informations spécifiques à la plateforme cible. La figure 6.18 illustre un extrait de code de l'exécutable du générateur «genSim ». Il expose l'association d'une configuration de type « ICC » aux périphériques de

contrôle « ICC » dans le « SIM ». Ces configurations sont encapsulées dans des balises « simConfiguration » (Figure. 6.18 lignes 7-16).

```
1  if (cc.direction.equals("in") ) {
2      Vector iccConfigs = new Vector();
3      iccConfigs = simConfigs4MAGALI.get_icc_config(cc.getName()) ;
4      for (int k=0 ; k<iccConfigs.size() ;k++){
5          ICCConfig icc_config = new ICCConfig();
6          icc_config = (ICCConfig)iccConfigs.get(k);
7          sim_output += "<simConfiguration  type = \"ICC\">\n" ;
8          sim_output += "<parameter name= \"" + parameters[0] + "\"  value= \""
9                         +icc_config.channel+ "\"/> \n"  ;
10         sim_output += "<parameter name= \""+ parameters[1] +"\"  value= \""
11                        +icc_config.num_occ+ "\"/> \n"  ;
12         sim_output += "<parameter name= \""+ parameters[2] +""\"  value= \""
13                        +icc_config.sel_credit+ "\"/> \n"  ;
14         sim_output += "<parameter name= \""+ parameters[3] +""\"  value= \""
15                        +icc_config.path_to_target.toString()+ "\"/> \n"  ;
16         sim_output += "</simConfiguration>\n";
17     }
18 }
```

Figure 6.18. Extrait du générateur « genSim » pour les composants « ICC »

Une fois généré, le modèle SIM fournit une représentation détaillée et orientée pour la plateforme Magali. En effet, des configurations sont attachées à chaque composant matériel de la plateforme. Elles sont encapsulées dans des balises XML dont le nom est « simConfiguration ».

Dans le reste de cette section, nous détaillons les informations spécifiques à Magali qui apparaissent dans les balises « simConfiguration ». Nous commençons par les périphériques de contrôle (ICC et OCC), puis les composants de calcul IPs, les composants SME et nous finissons par les interfaces réseau. On distingue plusieurs types de configurations dans le « SIM », selon le composant matériel de la plateforme. Leur classification est illustrée dans le tableau suivant :

Composant Matériel	Type de la configuration (SimConfiguration)
IP	CORE, SME_CORE, STR, LOAD
Interface Réseau	IP_CTX, ICC_CTX, OCC_CTX
OCC	OCC, OCC_SME
ICC	ICC

Tableau 1 : Classification des configurations dans le modèle SIM pour les composants de la plateforme Magali

6.4.2.2. Les périphériques de contrôle (ICC et OCC)

Dans le modèle SIM, les balises « SimConfiguration » sont ajoutées aux périphériques de contrôle précédemment décrits dans le modèle GIM. Nous distinguons les configurations de type « ICC » et « OCC » respectivement pour les périphériques « ICC » et « OCC ». Chacun peut disposer de plusieurs configurations.

Une configuration de type « OCC » (Figure. 6.19 lignes 1,8) est définie par quatre paramètres principaux :

- « channel » : canal virtuel utilisé pour envoyer les paquets de crédits (Figure. 6.19 lignes 2, 9).
- « num_icc » : identifiant du périphérique « ICC » correspondant à l'IP destinataire qui reçoit les données (Figure. 6.19 lignes 3, 10).
- « sel_credit » : Le mécanisme de compteur de crédits est utilisé pour le transfert des données entre les différents IPs à travers les périphériques de contrôle. L' « ICC » correspondant au composant destinataire envoie des crédits à l'émetteur en fonction de ses capacités de stockage de données. L'IP émetteur envoie des données selon les informations de crédit reçues. Il arrive dans Magali qu'un IP envoie des données à plusieurs autres IPs à travers le même périphérique OCC (Figure. 6.19). Un identificateur « sel_credit » dans la configuration de l' « OCC », est ainsi associé à chaque communication. Sa valeur est déduite depuis le modèle GIM, à travers une analyse des chemins de communication physiques.
- « path_to_target » : ce paramètre définit d'une manière explicite le chemin à prendre à travers le réseau pour réaliser une communication depuis l'IP émetteur vers le destinataire (Figure. 6.19 lignes 5-6, 12-13).

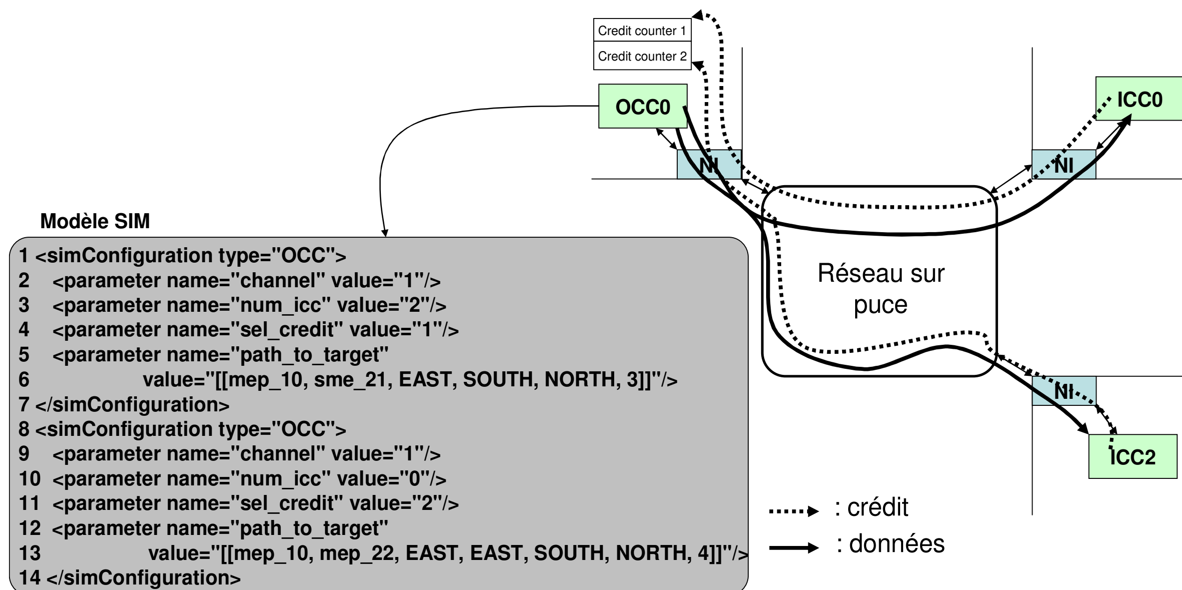


Figure 6.19. Exemple de configuration de type « OCC » dans le modèle SIM

Une configuration de type « ICC » associée au périphérique de contrôle « ICC » est décrite de la même manière que pour un « OCC » à quelques différences prés. Celle de l'« ICC » dispose d'un paramètre additionnel nommé « num_occ » qui identifie le périphérique « OCC » de l'IP source qui envoie les données.

Les configurations de type « ICC » et « OCC » sont utilisées pour la génération des fichiers de configuration de communication à savoir les fichiers « icc.cfg » et « occ.cgf ».

6.4.2.3. Les composants IPs de calcul

Dans le modèle « SIM », les éléments « SimConfiguration » sont ajoutés aux composants IPs de la plateforme Magali. Nous distinguons trois types de configurations : « CORE », « LOAD » et « STR ».

Les configurations de type « CORE » traduisent la description de l'API HAL des IPs. Celles de type « LOAD » définissent le chemin de communication entre le CPU principal et l'IP concerné. Ce chemin est utilisé par le processeur d'orchestration pour communiquer avec les IPs. Les configurations de type « STR » sont définies à partir des paramètres architecturaux des IPs. Elles contiennent des informations comme le nombre de FIFO en entrée et en sortie de l'IP ainsi que leurs tailles. Elles incluent des données sur la taille de la zone mémoire qui stocke les microprogrammes d'ordonnancement des configurations de communication et de calcul. Ces microprogrammes sont appelés des configurations de contexte et nous les associons aux composants de type « NI ».

L'ensemble de ces configurations servent dans une prochaine étape à générer les fichiers de configuration « core.cfg », « ip.load » et « ip.str ».

6.4.2.4. Les composants SME

Les éléments « SimConfiguration » dans le modèle SIM, pour les composants SME et ses périphériques OCC, décrivent une syntaxe particulière des configurations. Ces dernières sont de type « OCC_SME » et « SME_CORE ». Celles de type « SME_CORE » référencent les buffers du SME et définissent la mise en correspondance entre les canaux physiques et les canaux logiques du composant.

Celles de type « OCC_SME » décrivent quatre sorties, contrairement à un « OCC » classique qui dispose d'une seule. La figure 6.20 illustre un exemple d'élément « simConfiguration » de type « OCC_SME », correspondant à un périphérique « OCC » interconnecté au composant « sme_21 ». Il dispose de deux balises « output ». Chacune modélise une sortie de l' « OCC », identifiée par un numéro (balise « identifier » : 0, 1). Elle décrit les éléments de base d'une configuration de type « OCC », à savoir le numéro du périphérique « ICC » correspondant à l'IP qui reçoit les données, l'identifiant du compteur de crédits, le chemin de communication à travers le réseau et le nombre de données à transférer.

Dans la figure 6.20, nous générons deux compteurs de crédits différents pour chaque sortie de l' « OCC » afin de garantir la cohérence des communications sur le réseau.

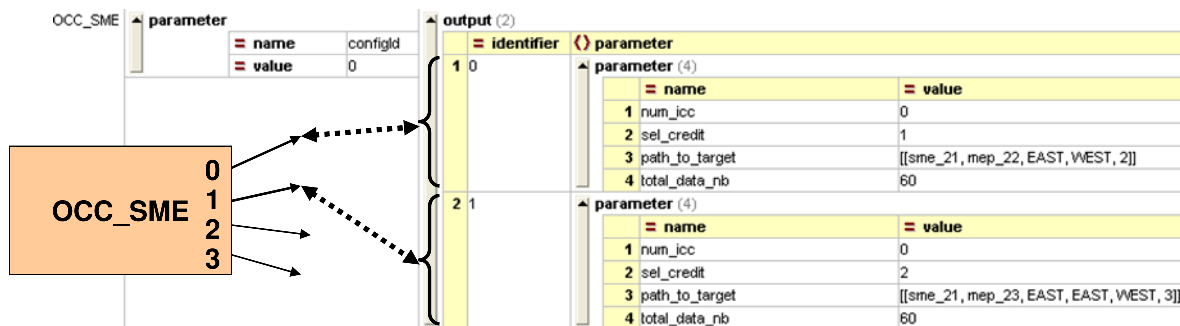


Figure 6.20. Exemple d'une configuration « simConfiguration » de type « OCC_SME »

Les configurations « simConfiguration » de type « SME_CORE » et « OCC_SME » permettent la génération respective des fichiers de configuration « core.cfg » et « occ.cfg » (représentés en Figures. 6.16.f et 6.16.g).

6.4.2.5. Les interfaces réseau (NI)

Les éléments « SimConfiguration » ajoutés aux composant de type « NI » sont de type contexte. Ils contiennent des configurations d'ordonnancement du calcul (« IP_CTX ») et des communications (« ICC_CTX » et « OCC_CTX »). Ils référencent et orchestrent les différentes configurations précédemment décrits : « OCC », « OCC_SME », « ICC », « CORE » et « SME_CORE ».

Les configurations de type contexte sont générées à partir des informations provenant des canaux de communication logiciels et en particulier depuis les arbitres, qui sont associés aux périphériques de contrôle. Les scénarii d'ordonnancement des communications définis dans les matrices d'entrée et de sortie des canaux de communication sont traduits en des configurations de type « ICC_CTX » et « OCC_CTX ».

Ces configurations de contexte, dans le modèle SIM, sont utilisées pour générer des microprogrammes d'ordonnancement des communications et des tâches, dans les fichiers « ctx.cfg ». Ces derniers sont physiquement exécutés par les unités CCC, réparties sur les différents IPs de la plateforme Magali.

A ce stade, Nous avons réussi à générer le modèle « GIM » qui est indépendant de Magali. Nous avons concrétisé son raffinement vers le « SIM ». Dans ce dernier, nous avons réussi à faire apparaitre plusieurs types de configurations que nous avons associés aux différentes ressources de Magali. La prochaine étape est l'exploitation du modèle « SIM » pour la génération du code de configuration.

6.5. Génération du code de configuration pour le déploiement de l'application 3GPP-LTE sur la plateforme Magali

Plusieurs types de fichiers de configuration sont à générer pour déployer l'application 3GPP-LTE sur la plateforme Magali :

- Les catalogues des configurations élémentaires de communication (icc.cfg et occ.cfg) et de calcul (core.cfg, ip.load, ip.str)
- Les configurations de contexte (ctx.cfg) qui ordonnancent les configurations élémentaires de calcul et de communication
- Les scénarii d'exécution de l'application qui décrivent son comportement global à travers l'orchestration et la synchronisation des configurations citées précédemment (ip.sno, cpu.sno)

La figure 6.21 illustre la répartition de ces différentes configurations dans la structure en couches du code de configuration.

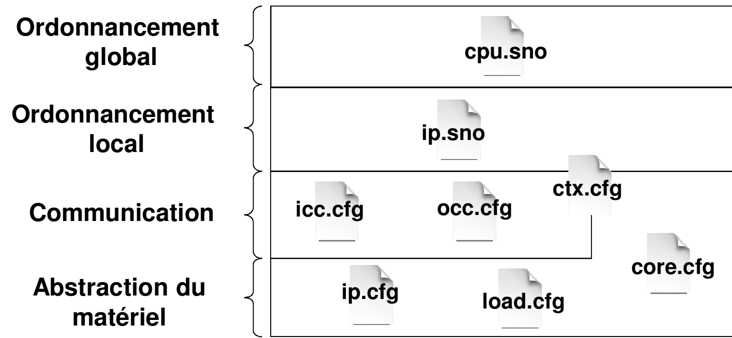


Figure 6.21. Répartition des fichiers de configuration pour programmer la plateforme Magali sur l'organisation en couches du code de configuration

La génération des fichiers de configuration passe par l'exécution de l'élément IP-XACT « generator » intitulé « genConfigs » présenté dans le paragraphe 5.6.1. Ce dernier prend comme paramètres d'entrée le modèle intermédiaire « SIM ».

6.5.1. Construction des catalogues des configurations de communication et de calcul

Les configurations élémentaires de communication et de calcul sont définies dans les fichiers « occ.cfg », « icc.cfg », « core.cfg », « ip.load » et « ip.str ». Nous les générons pour chaque nœud de la plateforme Magali, y compris les nœuds contenant des composants SME.

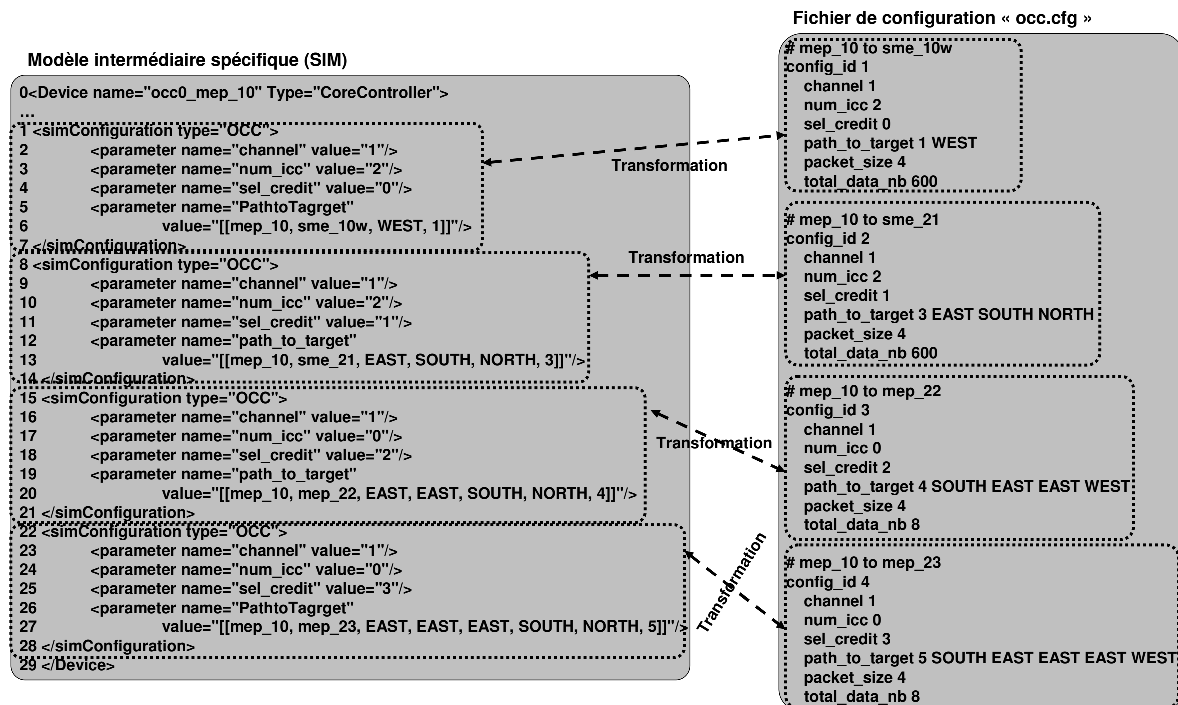


Figure 6.22. Exemple de génération du fichier de configuration « occ.cfg » depuis le modèle SIM

Des transformations sont appliquées sur les balises « simConfiguration » du modèle SIM, qui décrivent en XML les configurations de type « OCC », « OCC_SME », « ICC »,

«CORE », « SME_CORE », « LOAD » et « STR » pour produire les fichiers souhaités. Nous rappelons qu'un des objectifs de la proposition du modèle intermédiaire SIM est de fournir une représentation spécifique à la plateforme cible et proche du code final.

La figure 6.22 schématise la génération du fichier de configuration « occ.cfg », du périphérique « occ0_mep10 », à partir des configurations de type « OCC » décrites dans le modèle « SIM ». Ces dernières sont rassemblées dans un seul fichier appelé « occ.cfg ». Nous générons, pour chacune des configurations « OCC » provenant du « SIM », un identificateur (paramètre « config_id »).

La transformation (du modèle « SIM vers le code de configuration) est simple à réaliser. En effet, tous les paramètres des configurations sont présents dans le modèle « SIM » à savoir l'identifiant du « ICC » source, le chemin des données à transférer à travers le réseau sur puce, etc.

6.5.2. Construction des configurations de contexte

Les configurations de contexte sont définies dans un fichier « ctx.cfg » pour chaque nœud de la plateforme Magali.

Ce fichier contient des microprogrammes qui ordonnancent les configurations élémentaires de calcul et de communication, précédemment générées dans les fichiers « icc.cfg », « occ.cfg » et « core.cfg ». Ces microprogrammes référencent les configurations élémentaires par leurs identificateurs. Des transformations sont appliquées sur les balises « simConfiguration » du modèle SIM, qui décrivent en XML les configurations de type « IP_CTX », « ICC_CTX » et « OCC_CTX », pour produire les fichiers « ctx.cfg ».

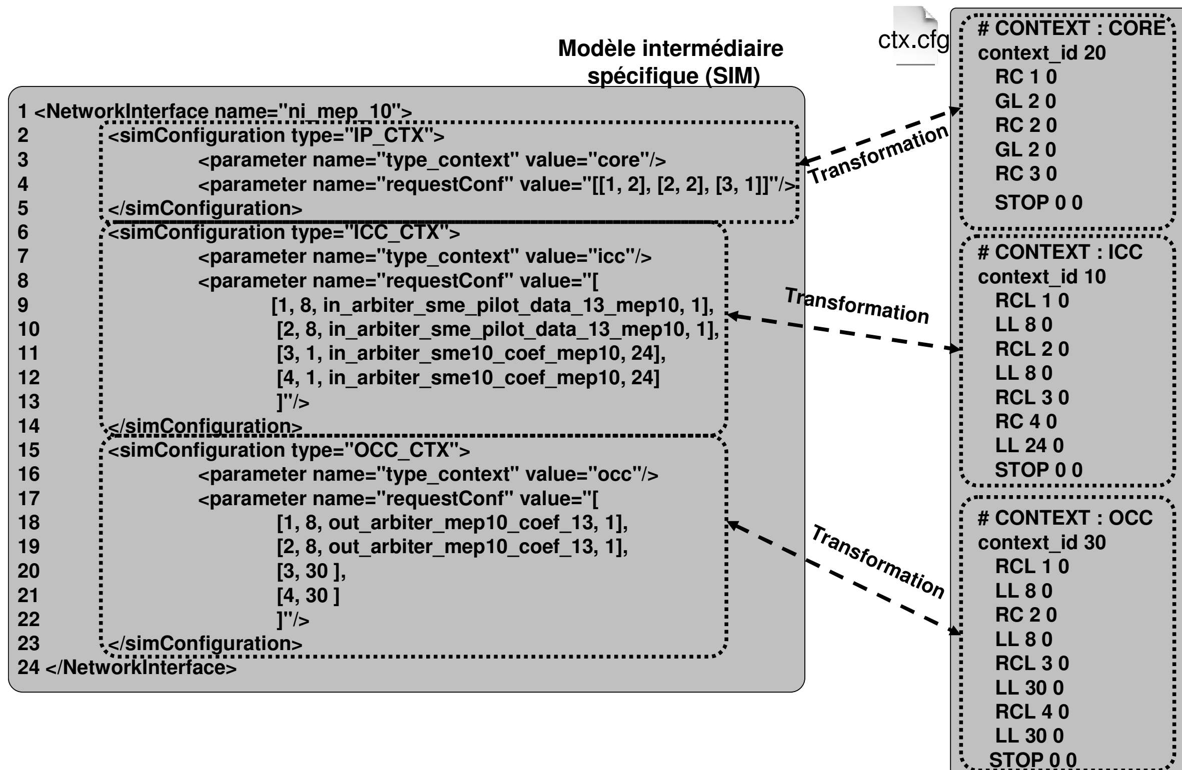


Figure 6.23. Exemple de génération du fichier de configuration « ctx.cfg » depuis le modèle SIM

La figure 6.23 illustre un exemple de passage depuis les configurations de type « IP_CTX », « ICC_CTX » et « OCC_CTX » du modèle SIM, vers le fichier « ctx.cfg ». Ce dernier les rassemble en un ensemble de microprogrammes qui utilisent des primitives exécutables par l'unité CCC (RC : Request Configuration, GL : Global Loop, etc.). Cette transformation est également simple à réaliser puisque toutes les informations nécessaires sont fournies par le modèle « SIM », à savoir l'ordonnancement des exécutions des différentes configurations, le type du microprogramme à générer, etc.

Le microprogramme dont l'identifiant est égal à « 10 » (context_id 10) enchaîne les configurations de lecture des données du fichier « icc.cfg ». Il commence par exécuter la configuration dont l'identifiant est égal à « 1 » grâce à la primitive « RCL : Request Configuration Loop). Elle est exécutée 8 fois (LL (Local Loop) 8 0) puis la configuration « ICC » dont l'identifiant est égal « 2 » est exécutée 8 fois également.

Ensuite les configurations dont les identifiants sont « 3 » et « 4 » sont exécutées 24 fois d'une manière alternée. Dans cet exemple, nous avons généré un microprogramme pour les communications entrantes (context_id 10), un pour les communications sortantes (context_id 30) et un pour l'ordonnancement des exécutions des tâches sur l'IP (context_id 20).

Les fichiers que nous générons couvrent les couches « Abstraction du matériel », « communication » et une partie de la couche « Ordonnancement local » de la structure en couches du code de configuration. Le nombre total des fichiers de configuration est d'environ 80. Nous générons à ce jour environ 65. Les autres sont ajoutés manuellement pour effectuer la simulation de l'application 3GPP-LTE sur la plateforme Magali.

6.5.3. Code de configuration non généré

Nous n'avons pas réussi à générer le troisième type de fichiers de configuration nécessaires au déploiement de l'application 3GPP-LTE sur la plateforme Magali. Il s'agit des fichiers de scénarii globaux et locaux. Nous n'avons pas également réussi à traiter les flots de données applicatives dynamiques. Nous détaillons ces deux points dans la suite.

6.5.3.1. Les scénarii globaux et locaux

Dans l'étude de cas présentée, nous n'avons pas pu générer les fichiers « ip.sno » pour chaque IP et « cpu.sno » qui représente le comportement global de l'application. Ces deux fichiers implantent les couches « Ordonnancement global » et « Ordonnancement local » de la structure du code de configuration.

En effet, nous avons besoin de représenter dans le modèle « SIM », des informations sur les adresses des configurations de communication et de contexte dans les zones mémoires du CCC. Les adresses d'interruptions de chaque IP doivent également être représentées. Cela est important pour initialiser les gestionnaires de configuration et de contexte, qui sont des blocs de base dans le fonctionnement du CCC. En effet, Le scénario global utilise des primitives telles que : SND_ITM_CFG (send interrupt manager configuration), SND_CFM_CFG (send configuration for configuration manager), et la primitive « SND_CTX_INSTR » pour le transfert des instructions des configurations de contexte dans

la mémoire de l'interface réseau. Ces informations de bas niveau ne sont pas encore prises en considération et cela nécessiterait un peu plus de développement. Ainsi, une intervention manuelle d'un développeur pour initialiser les gestionnaires de configuration est pour le moment indispensable. Elle est nécessaire également pour séquencer les opérations de chargement des configurations, d'envoi et de réception des messages d'interruption, etc.

6.5.3.2. Prise en considération des applications dynamiques

Nous avons déployé dans ce chapitre une application de télécommunication 3GPP-LTE avec des flots de données fixes et connus dès le départ, ce qui nous a permis de générer les différentes configurations de communication d'une manière précise.

Il se trouve que ces applications pourraient disposer de flots de données dynamiques. La représentation APDL supporte la description de tels flots. La plateforme Magali supporte également le déploiement de ce type d'application en définissant des microprogrammes (configurations de contexte) paramétrables par des variables. Nous utilisons pour cela la primitive « LLi » qui prend en argument des variables déclarées dans le fichier d'ordonnancement global « cpu.sno ». La valeur de la variable n'est pas fixe et change selon le mode de transfert de données de l'application.

Afin de supporter une telle variabilité dans les échanges des données, nous devons fournir toutes les configurations possibles dès le départ. Le nombre de valeurs que peuvent prendre les variables lors de l'exécution de l'application est limité. Sinon la génération de nouvelles configurations est nécessaire à chaque fois qu'une variable change de valeur. Pour le moment, nos outils de génération de code de configuration ne prennent pas en compte cet aspect dynamique des applications dans leurs déploiements sur la plateforme Magali. Il s'agit d'une perspective du travail présenté dans ce mémoire.

6.6. Simulation des fichiers de configuration

Nous rappelons qu'un modèle de simulation en SystemC est généré à partir d'outils et d'une représentation RTL de la plateforme, développés par le CEA-LETI et MDS. Les différents composants (IPs, interfaces réseau) du modèle SystemC extraient les informations depuis le code de configuration pour exécuter l'application. Il s'agit d'une méthode pour valider les fichiers de configuration générés en analysant les résultats attendus de simulation.

La figure 6.24 illustre l'extraction des données par le modèle de simulation depuis les fichiers de configuration (Figure. 6.24.d). Un bout de code du fichier « ni_config.cpp », dans la figure 6.24.d, montre comment les configurations de type « OCC » sont extraites à travers la méthode « init_config » (ligne 1). Cette dernière ouvre le fichier « occ.cfg » (lignes 19-20) et sauvegarde son contenu dans les variables locales déclarées dans les lignes 8-15.

L'exécution de ce code pour la simulation du composant IP « recgen_00w » est illustrée dans la figure 6.24.e. Nous pouvons voir que le modèle de simulation détecte qu'il n'existe pas de configurations de type « ICC » et qu'il existe une unique configuration de type « OCC » extraite depuis le fichier « ./appli/recgen_00w/occ.cfg ».

La même méthode est appliquée pour extraire toutes les autres configurations de l'IP (.sno, ctx.cfg, etc.) (Figure. 6.24.a).

Une fois que les fichiers de configuration sont explorés, et que les configurations retrouvées sont affectées aux paramètres des composants du modèle de simulation, le déploiement de l'application peut être lancé et simulé. La figure 6.24.b illustre les échanges de données effectuées par l'IP « recgen_00w ». Il est indiqué sur la figure que cet IP a envoyé 7600 données (DS (Data Sent) : 7600) et qu'il a reçu 7600 crédits (CR (Credit Received) : 7600). L'IP n'a pas reçu de données, ni envoyé de crédit ce qui correspond exactement au comportement de la tâche applicative que nous avons associée à cet IP. Des estimations du temps d'exécution de l'application sont également fournies par le modèle de simulation (Figure. 6.24.c).

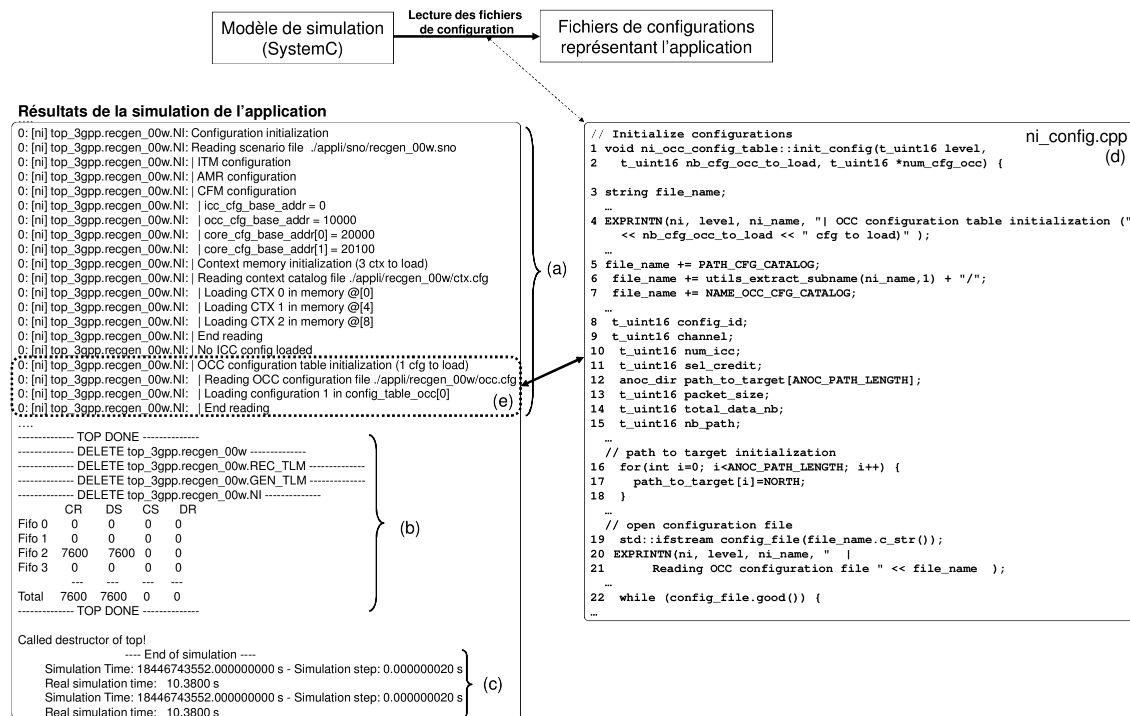


Figure 6.24. Simulation des fichiers de configuration de l'application 3GPP-LTE

A ce stade, nous avons réussi à valider le code de configuration généré à travers le modèle de simulation écrit en SystemC. L'intérêt de notre outil de génération automatique de code de configuration est d'obtenir rapidement un nouveau jeu de configurations si l'on modifie le partitionnement (placer une tâche sur un autre IP par exemple) ou si l'on ajoute de nouvelles ressources matérielles.

6.7. Conclusion

Nous avons réussi, à travers cette étude de cas, de fournir un premier outil de génération automatique du code de configuration pour la plateforme Magali. Le développement manuel des fichiers de configuration pour le déploiement de l'application 3GPP-LTE sur Magali prenait entre quelques semaines jusqu'à quelques mois. Il s'agit s'une

estimation qui nous a été communiquée par le CEA-LETI. En appliquant notre flot, ce temps est réduit pour devenir entre une et deux semaines.

Chapitre 7. Vers un flot commun de génération de code

7.1. Rappel de la problématique 115
7.2. Présentation de l'environnement « APES » pour le développement du code logiciel exécutable par des processeurs 116
7.3. Définition d'un flot commun de génération de code 118
7.4. Modèle de représentation ARDL pour les architectures multiprocesseurs...................... 118
7.5. Description des flots de données KPN avec la représentation APDL................................ 120
7.6. Partitionnement des ressources 120
7.7. Modèles intermédiaires... 121
7.8. Application du flot commun pour programmer les architectures multiprocesseurs....... 122
7.9. Conclusion 125

7.1. Rappel de la problématique

Ce chapitre s'intéresse aux possibilités de généraliser le flot de génération de code de configuration (Figure. 7.1). En effet, nous nous attendons à voir émerger des architectures qui intègrent des processeurs pour lesquels il faudra générer le code binaire exécutable d'une part, et des IPs autour d'un NoC avec des interfaces réseau configurables, pour lesquels il faudra générer le code de configuration d'autre part. Il nous faut donc réfléchir à un environnement commun pour générer ces deux types de code.

Un exemple de flot classique de génération de code exécutable pour les processeurs, basé sur l'environnement de développement logiciel APES (Figure. 7.1.a) [Apes] [Gue09], existe dans l'équipe SLS du laboratoire TIMA. Il a été développé par X. Guérin, K. Popovici et A. Chagoya dans le cadre du projet SHAPES [Spo07]. Ce flot cherche à déployer des applications orientées flots de données décrites en KPN sur des architectures multiprocesseurs. Nous cherchons dans ce chapitre à intégrer ce flot avec celui que nous avons proposé pour la génération du code de configuration afin de généraliser ce dernier.

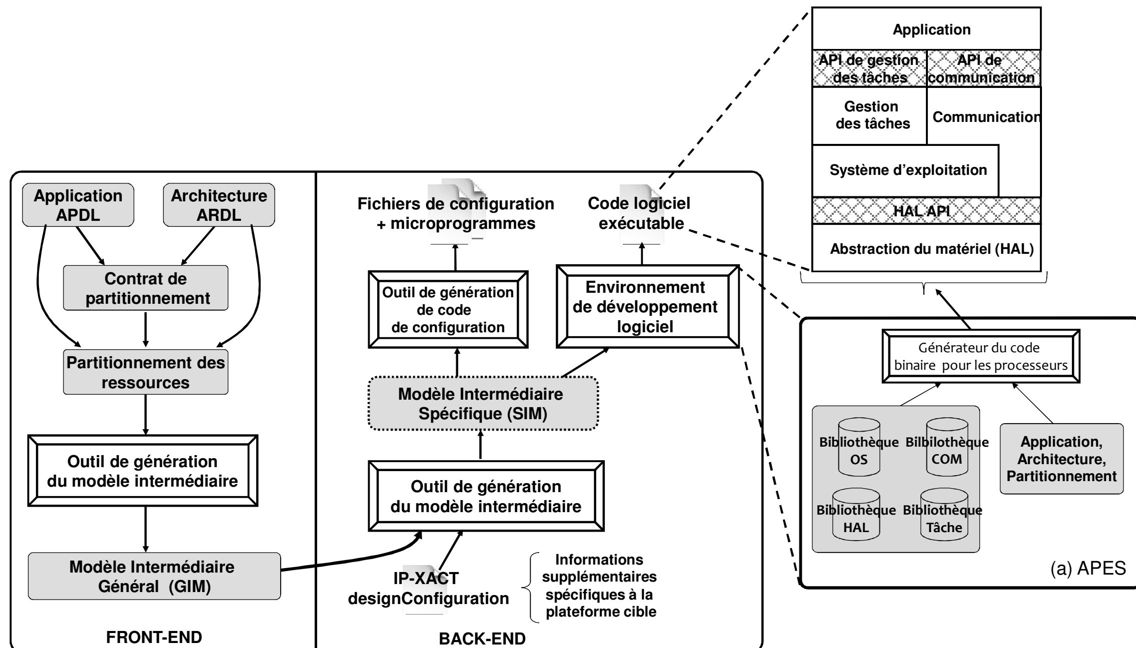


Figure 7.1. Vers un flot commun de génération de code

Ainsi, nous disposons de deux flots de génération de code différents qui ciblent deux types d'architectures matérielles différentes. Chacun dispose de modèles d'entrée ainsi que d'outils de génération et de transformation spécifiques. Nous les unifions dans ce chapitre afin de proposer un seul flot commun (Figure. 7.1). Cela suppose que nous disposons de représentations communes des descriptions des applications, des divers types d'architectures et du partitionnement des ressources.

7.2. Présentation de l'environnement « APES » pour le développement du code logiciel exécutable par des processeurs

L'environnement « APES » permet de développer facilement le code logiciel au format binaire pour les plateformes multiprocesseurs. Il s'agit d'un environnement configurable à base de modules qui permet le choix, l'intégration, l'assemblage et la compilation des composants logiciels (systèmes d'exploitation, librairies de communication, pilotes de périphériques, etc.). Nous utilisons cet environnement pour générer le code logiciel organisé selon la structure en couches détaillée dans le chapitre 2 et représentée en Fig.2.5.

L'utilisation de cet environnement passe par la création de plusieurs entrées :

1) Les descripteurs des modules logiciels : Un fichier « component.xml » est associé à chaque module logiciel. Ce dernier peut être un pilote, un sous module d'un système d'exploitation, une bibliothèque de communication ou même une application logicielle, etc. Le fichier « component.xml » d'une application logicielle contient des références vers d'autres modules nécessaires (système d'exploitation par exemple) pour compiler et générer le code logiciel final. Il s'agit d'une description des dépendances de l'application. Les modules référencés sont : la liste des pilotes à utiliser, la bibliothèque de communication, la bibliothèque de gestion des tâches, etc.

2) Le fichier « install.sh » contient des paramètres qui définissent la chaine de compilation et d'outils indispensables à l'assemblage et à la compilation.

3) Le plan mémoire : Le fichier « ldscript », créé pour chaque processeur de l'architecture matérielle, définit les sections mémoire où le logiciel binaire est réparti. Les plus importantes sections sont celles qui correspondent au système d'exploitation (.os_config) et à la couche d'abstraction du matériel (.hal) (Figure. 7.2.a). A travers ce fichier, nous configurons le système d'exploitation DNA [Gue09] [Gar09] utilisé dans l'environnement APES. Les pilotes de communication ou de composants matériels, et en particulier les périphériques, sont également choisis à ce niveau.

Exemple : En utilisant l'environnement « APES », nous développons le code logiciel de l'application « Mjpeg ». Pour cela, un module « mjpegApplication » (Figure. 7.2.A) est créé.

Le code de l'application est sauvegardé dans les répertoires « Headers » et « Sources ». Le fichier « install.sh » (Figure. 7.2.c) décrit la chaine d'outils (compilateur, éditeur de liens). Le fichier « component.xml » (Figure. 7.2.b) identifie les dépendances entre l'application et d'autres modules logiciels fournis dans l'environnement « APES ».
Supposons que cette application est à simuler sur une plateforme décrite en « SoCLib ». Il s'agit d'une bibliothèque qui permet de définir des modèles de simulation SystemC pour des architectures multiprocesseurs en vue d'un éventuel prototypage virtuel. Les architectures multiprocesseurs sont simulées au niveau CABA (Cycle Accurate Bit Accurate). Le

module logiciel de l'application « Mjpeg » doit référencer ceux des pilotes des composants « SoCLib ». Cette liaison est réalisée dans le fichier «component.xml» (les pilotes des composants périphériques « SoCLib » du module « SoclibPlatformDriver » (Figure. 7.2.B) sont liés au module applicatif « MjpegApplication »).

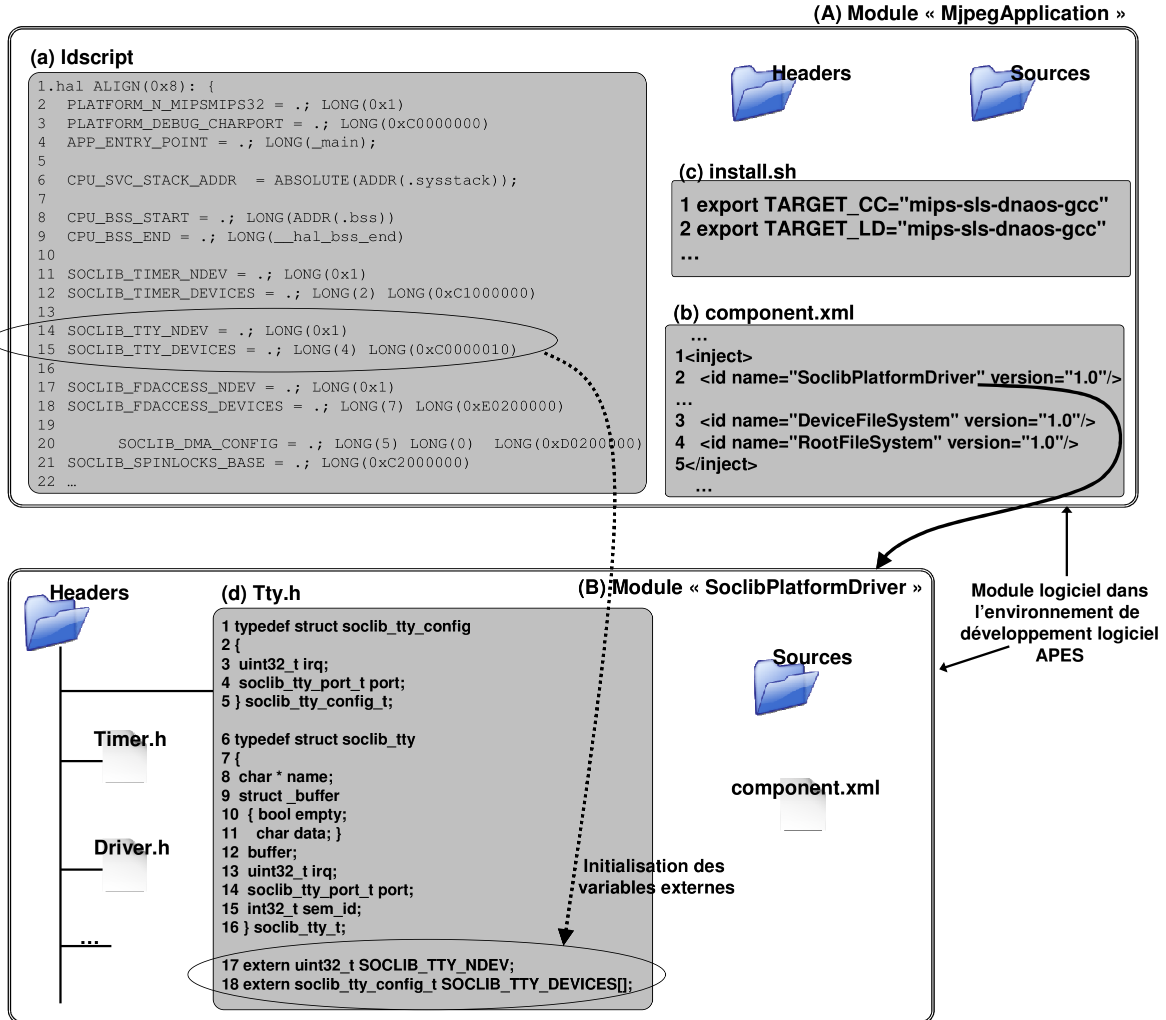


Figure 7.2. Développement du module « MjpegApplication » dans l'environnement « APES »

La section mémoire « .hal » du fichier « ldscript » (Figure. 7.2.a) configure les pilotes, dont le code source se trouve au niveau du module logiciel «SoclibPlatformDriver ». Un bout de code de l'interface du pilote du périphérique « tty » est illustré dans la figure 7.2.d. Les variables externes (Figure. 7.2.d lignes 17 et 18) sont initialisées par les valeurs éditées dans le fichier « ldscript », au niveau de la section mémoire « .hal » (Figure. 7.2.a lignes 14, 15).

A ce stade, nous avons présenté l'environnement de développement du code logiciel APES. Dans la suite, nous utilisons notre flot de génération de code de configuration pour générer les fichiers d'entrée de l'environnement APES. Cela permet de généraliser notre

approche pour supporter la génération du code de configuration d'un coté, et la génération des entrées vers les environnements de développement du code logiciel pour les processeurs, d'un autre coté.

7.3. Définition d'un flot commun de génération de code

Dans ce paragraphe, nous présentons notre proposition d'un flot commun pour la génération de code, fondée sur des descriptions d'architectures et d'applications de haut niveau.

Nous réutilisons les représentations APDL et ARDL pour la description respective des applications orientées flots de données et des architectures matérielles. Nous réutilisons également le contrat de partitionnement pour diversifier les possibilités de portage.

Le choix de notre approche en deux grandes étapes « Front-End/Back-End » s'impose si nous ciblons plusieurs familles de plateforme. En effet, la première phase « Front-End » est indépendante des plateformes et représente un point d'entrée partagé. Cette phase donne un caractère commun au flot de génération de code.

La partie « Back-End » prend en considération les caractéristiques de la plateforme cible et permet d'intégrer les environnements de développement du code logiciel et/ou de simulation. Les modèles intermédiaires « GIM » et « SIM » permettent le raffinement successif des informations liées aux ressources matérielles et logicielles. Nous avons vu dans le chapitre 6 le mode de passage à partir du modèle « SIM », grâce aux informations décrites dans les balises « simConfiguration », vers le code de configuration final (les fichiers de configuration et les microprogrammes). Nous cherchons à produire un modèle « SIM » exploitable par l'environnement APES pour générer le code logiciel binaire et le code de simulation. Il s'agit d'un modèle « SIM » à partir duquel nous générons les fichiers « component.xml », « install.sh » et « ldscript ».

Dans la suite, nous présentons l'approche de généralisation du flot de génération de code de configuration. Cette dernière a nécessité quelques modifications dans la représentation ARDL.

7.4. Modèle de représentation ARDL pour les architectures multiprocesseurs

La représentation ARDL que nous avons définie est spécifique aux architectures à base d'IPs interconnectés à un réseau sur puce configurable. Pour modéliser les architectures multiprocesseurs, il manque à ARDL, la possibilité de définir plusieurs types de périphériques et de supports de communication, à savoir les bus.

Nous avons généralisé ARDL pour représenter les composants des architectures multiprocesseurs offrant ainsi plus de choix au niveau du support de communication, des composants périphériques et des processeurs. Une architecture écrite en ARDL décrit les ressources de calcul (les processeurs, les IP matériels configurables ou non configurables). Elle spécifie si le support de communication est un bus ou un réseau sur puce. Les ressources

de stockage, qui sont principalement des ressources mémoires, ainsi que d'autres ressources périphériques à savoir les DMA, TIMER, etc. sont également décrites.

La couche logicielle basse pour les architectures multiprocesseurs est décrite avec les balises « HALAPI » (Hardware Abstraction Layer). Ces dernières décrivent les services et les fonctionnalités fournies par les ressources matérielles (processeurs, périphériques, mémoires, etc.). Il peut s'agir de fonctions de changement de contexte des processeurs, de fonctions d'initialisation de l'horloge pour le périphérique TIMER ou de primitives de lecture et d'écriture pour les composants de mémorisation.

Les chemins de communication physiques sont décrits, pour les architectures multiprocesseurs, à travers les processeurs, les composants mémoires ainsi que certains périphériques tels que le DMA.

Exemple : La figure 7.3 illustre un exemple d'une architecture matérielle décrite avec une représentation ARDL dans l'environnement Magillem. Nous avons défini une bibliothèque de composants SoCLib (Figure. 7.3.a) avec une représentation ARDL. Un élément « Design » IP-XACT modélise la plateforme cible à partir des instances de ces composants.

L'architecture contient un processeur « MIPS », deux mémoires « ram », des périphériques de type « DMA », « timer », « fd », « fb », « tty » et « locks ». Tous ces composants sont interconnectés via un bus « vgmn ».

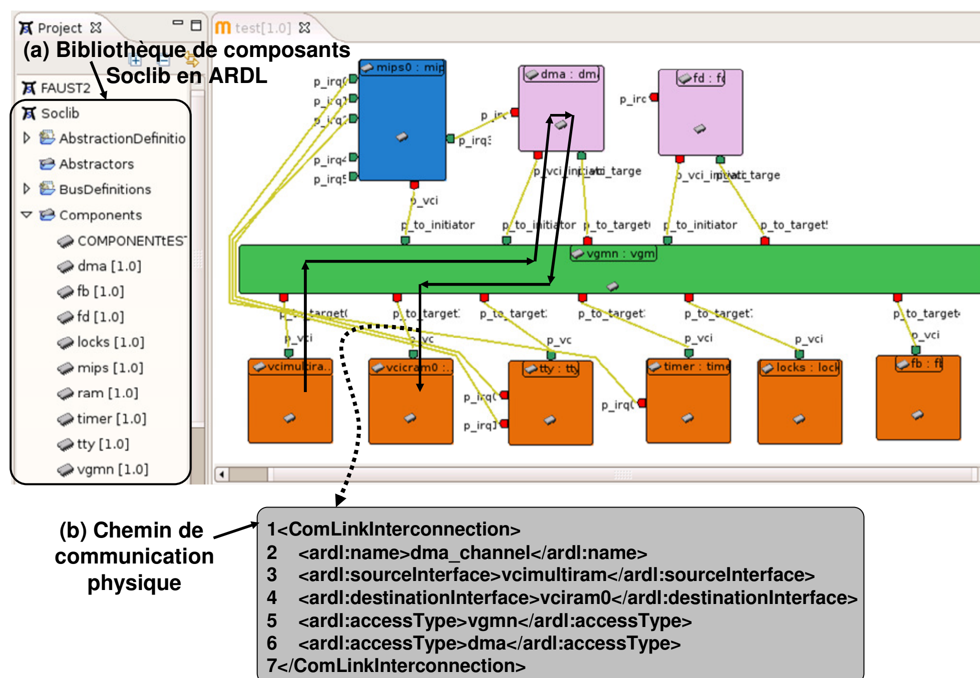


Figure 7.3. Représentation ARDL d'une architecture matérielle constitué de composants « SoCLib » virtuels

La figure 7.3.b illustre un chemin de communication partant d'une mémoire «vcimultiram » et arrivant à la mémoire « vciram0 », en passant à travers le bus « vgmn » et le

composant périphérique « DMA ». Ce chemin est associé à un canal de communication logiciel dans la phase de partitionnement. Cette information permet d'identifier le pilote du composant « DMA » comme étant un module à prendre en considération lors de la génération du code logiciel final.

7.5. Description des flots de données KPN avec la représentation APDL

La représentation APDL a été initialement proposée pour remédier aux inadaptations des langages de modélisation des applications qui respectent les modèles de calcul connus comme KPN et SDF. Ces inadaptations proviennent du fait que les représentations dans DOL [Thi07] et DSX [Dsx] ne permettent pas de représenter l'arbitrage des données échangées ; ce qui est possible avec la représentation APDL.
Nous pouvons tout de même représenter, grâce à APDL, des applications KPN ou SDF en définissant les canaux de communication à un seul port d'entrée et de sortie. Dans ce cas, il n'est plus nécessaire de modéliser un arbitre d'entrée ou de sortie.

Ainsi, nous avons choisi APDL comme modèle de représentation des applications dans le flot commun de génération de code. En effet, ce modèle de représentation est assez général et supporte la représentation de plusieurs schémas d'échanges de données entre les tâches applicatives.

Exemple : La figure 7.4 illustre une représentation APDL d'une application Mjpeg, que nous déployons sur la plateforme matérielle de la figure 7.3. Elle comporte trois tâches (fetch, compute et dispatch) et trois canaux de communication logiciels à une seule entrée et une seule sortie.

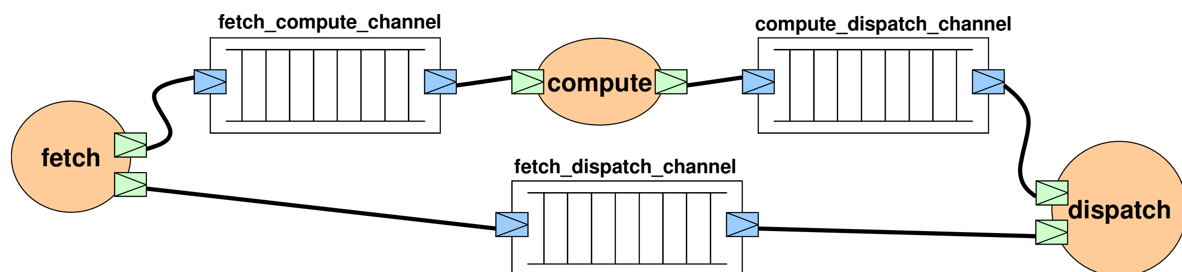


Figure 7.4. Représentation APDL d'une application Mjpeg

7.6. Partitionnement des ressources

Le partitionnement des ressources applicatives sur les composants des architectures multiprocesseurs est défini par l'association des tâches aux processeurs et des canaux de communication logiciels aux composants de mémorisation. Ces règles générales sont décrites dans le contrat de partitionnement.

Il est possible de décrire des règles spécifiques en mentionnant par exemple qu'un processeur particulier ne peut exécuter qu'une seule tâche (processeur mono-tâche) ou

plusieurs (processeur multi-tâches). Ces contraintes sont respectées dans la phase de partitionnement. Elles fournissent des informations utiles aux choix des différents modules du système d'exploitation (en particulier le type de l'ordonnanceur).

Cette technique de partitionnement a été validée pour la génération du code de configuration pour les interfaces réseaux configurables. Nous la réutilisons pour la génération de code logiciel des architectures multiprocesseurs.

Ainsi, le contrat de partitionnement permet de rompre la dépendance entre le type de l'architecture matérielle et le modèle de partitionnement, qu'on retrouve dans [Thi07] [Dsx]. Il joue le rôle d'un adaptateur entre l'architecture matérielle et le modèle de partitionnement qui lui est spécifique. Le modèle de partitionnement n'est pas défini au préalable, mais il est le résultat de l'application des règles de partitionnement.

7.7. Modèles intermédiaires

Nous réutilisons les deux modèles intermédiaires « GIM » et « SIM » dans le flot commun de génération de code. Le modèle « GIM » pour les architectures multiprocesseurs comporte les composants d'architecture (les processeurs, les périphériques, les composants de mémorisation, les chemins de communication physiques) auxquels sont associés les ressources applicatives (tâches, canaux de communication logiciels) selon le modèle de partitionnement élaboré.

Les informations spécifiques aux architectures multiprocesseurs, que nous apportons pour raffiner le modèle GIM, concernent le plan mémoire des composants de l'architecture, les espaces d'adressage des processeurs, les configurations des environnements de développement logiciel (choix des bibliothèques de communication, pilotes, système d'exploitation, chaîne de compilation, chaîne d'outils, etc.), etc. Ces informations sont décrites dans un élément « designConfiguration » étendu d'IP-XACT (présenté au paragraphe 5.4.3). Le modèle SIM est généré par un outil encapsulé dans un élément IP-XACT « generator » qui prend en entrée le modèle GIM et le « designConfiguration ».

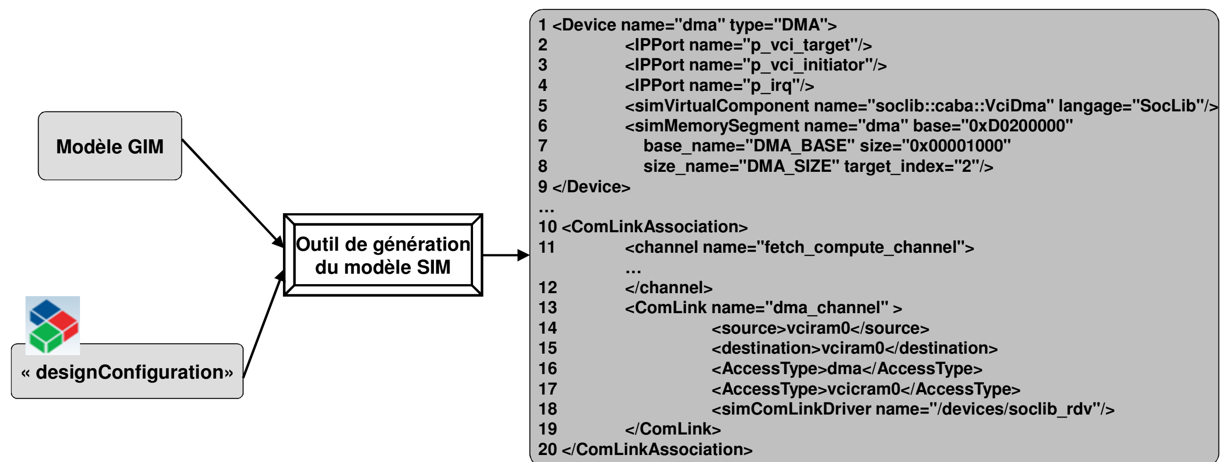


Figure 7.5. Extrait du modèle SIM pour la génération du code logiciel

Exemple : Le bout de code du modèle SIM de la figure 7.5 décrit les composants périphériques (DMA lignes 1-8) et les chemins de communication physiques (lignes 10-19). Les nouvelles informations qui apparaissent dans le modèle SIM pour le composant DMA, sont le nom du composant virtuel à instancier en simulation (soclib::caba::VciDam), la section mémoire correspondante à ce périphérique et son identifiant (le paramètre « target_index »). Cette dernière information est importante pour générer la table de partitionnement mémoire (MappingTable) spécifique à SoCLib.

Le chemin de communication « fetch_compute_channel » comporte le DMA, comme étant le composant qui réalise le transfert de données entre deux mémoires. Dans le modèle SIM, on génère une association du pilote « /devices/soclib_rdv » à ce chemin. Ce pilote sera intégré dans le logiciel binaire à développer et va réaliser la communication à base du DMA.

7.8. Application du flot commun pour programmer les architectures multiprocesseurs

7.8.1. Intégration des environnements de développement de code

L'intégration des environnements de développement de code se fait dans la partie « Back-End » du flot commun de génération de code. En effet, nous ne générons pas le logiciel exécutable en partant de zéro. Les différentes composantes du code final sont présentes dans l'environnement de développement logiciel. Ce dernier fournit des mécanismes d'assemblage et de composition des différents modules pour produire le code final.

Par intégration, nous faisons référence aux mécanismes de génération des entrées aux environnements de développements. Il s'agit des fichiers de configuration qui identifient les modules logiciels à assembler, les outils de compilation, etc.

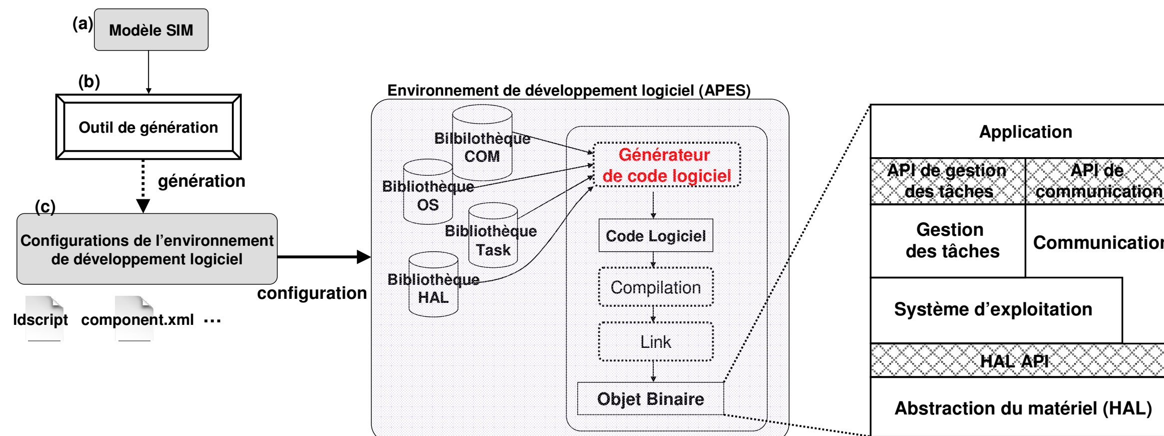


Figure 7.6. Intégration d'un environnement de développement logiciel

Les informations spécifiques à ces environnements sont représentées dans le modèle intermédiaire « SIM » (Figure. 7.6.a). Il s'agit de détails concernant le partitionnement mémoire des composants, le choix du système d'exploitation et de ses modules, les pilotes associés aux périphériques de l'architecture matérielle, les chaînes d'outils, les compilateurs, etc. Un outil extrait ces données depuis le modèle « SIM », comme illustré dans la figure 7.6.b, et génère les entrées des environnements de développement de code (Figure. 7.6.c). Nous présentons dans la suite comment nous avons intégré l'environnement de développement de code « APES » à la partie « Back-End » du flot commun.

7.8.2. Génération du code « SoCLib » et des fichiers configuration de l'environnement « APES »

À partir du modèle intermédiaire « SIM », nous générons le code SoCLib de simulation de l'architecture matérielle. Nous produisons le fichier « top.cpp » qui contient la définition de la table de partitionnement mémoire, les instances des composants virtuels, leurs signaux et leurs interconnexions. Nous générons également un fichier nommé « segmentation.h » qui englobe les déclarations des segments mémoires que nous associons à chacun des composants matériels de la plateforme.

La figure 7.7 illustre un exemple de transformation depuis le modèle « SIM » vers le modèle SoCLib de la plateforme. Le bout de code du modèle « SIM » de la figure 7.7.a décrit deux composants périphériques « tty » et « timer », auxquels sont ajoutées des informations spécifiques à SoCLib. Il s'agit du nom du composant virtuel à instancier (VciMultiTty et VciTimer), de l'adresse et de la taille du segment mémoire (base = « 0Xc1000000 », size = « 0x00000100 ») que nous associons à chacun de ces deux périphériques.

Les tailles des segments mémoires nous permettent de construire le fichier « segmentation.h » (Figure. 7.7.e) qui est inclus dans le modèle « top.cpp » (Figure. 7.7.b) de la plateforme. Les informations d'indexation de chaque composant (Figure. 7.7.a lignes 8 et 17) permettent de construire la table de partitionnement mémoire (MappingTable) spécifique à « SoCLib » (Figure. 7.7.b lignes 11-15). La description des interconnexions dans le modèle « SIM » (Figure. 7.7.a lignes 20-29) permettent de générer les signaux (Figure. 7.7.b lignes 17-19) ainsi que les interconnexions (Figure. 7.7.b lignes 23-25) entre tous les composants de la plateforme d'un coté et le bus de communication « vgmn » de l'autre coté.

Nous générons également, à partir du modèle « SIM », une partie des fichiers en entrée à l'environnement « APES » (les fichiers « component.xml », « install.sh » et « ldscript »). Une partie du fichier « component.xml » (Figure. 7.7.c), relative aux dépendances des modules logiciels applicatifs, est générée. Ces informations apparaissent dans les balises « simDriver » (Figure. 7.7.a lignes 9 et 18) du modèle « SIM », pour les composants périphériques, par exemple. Elles sont traduites dans le fichier « component.xml » à travers des appels des modules des pilotes des périphériques (Figure. 7.7.c ligne 2).

Une partie du fichier « ldscript » (Figure. 7.9.d) (une partie du segment « .hal ») est générée en se basant sur les informations des sections mémoires des pilotes des périphériques (Figure. 7.7.a lignes 5-8 et 14-17).

A ce stade, nous avons réussi à produire une partie des fichiers en entrée à l'environnement « APES » pour générer le code binaire exécutable par les processeurs. Nous sommes partis des modèles APDL et ARDL de l'application et de la plateforme et nous avons mis en œuvre notre flot de génération de code. A travers la génération de code pour l'environnement de développement APES, nous avons confirmé les possibilités de généralisation du flot de génération de code de configuration pour d'autres plateformes.

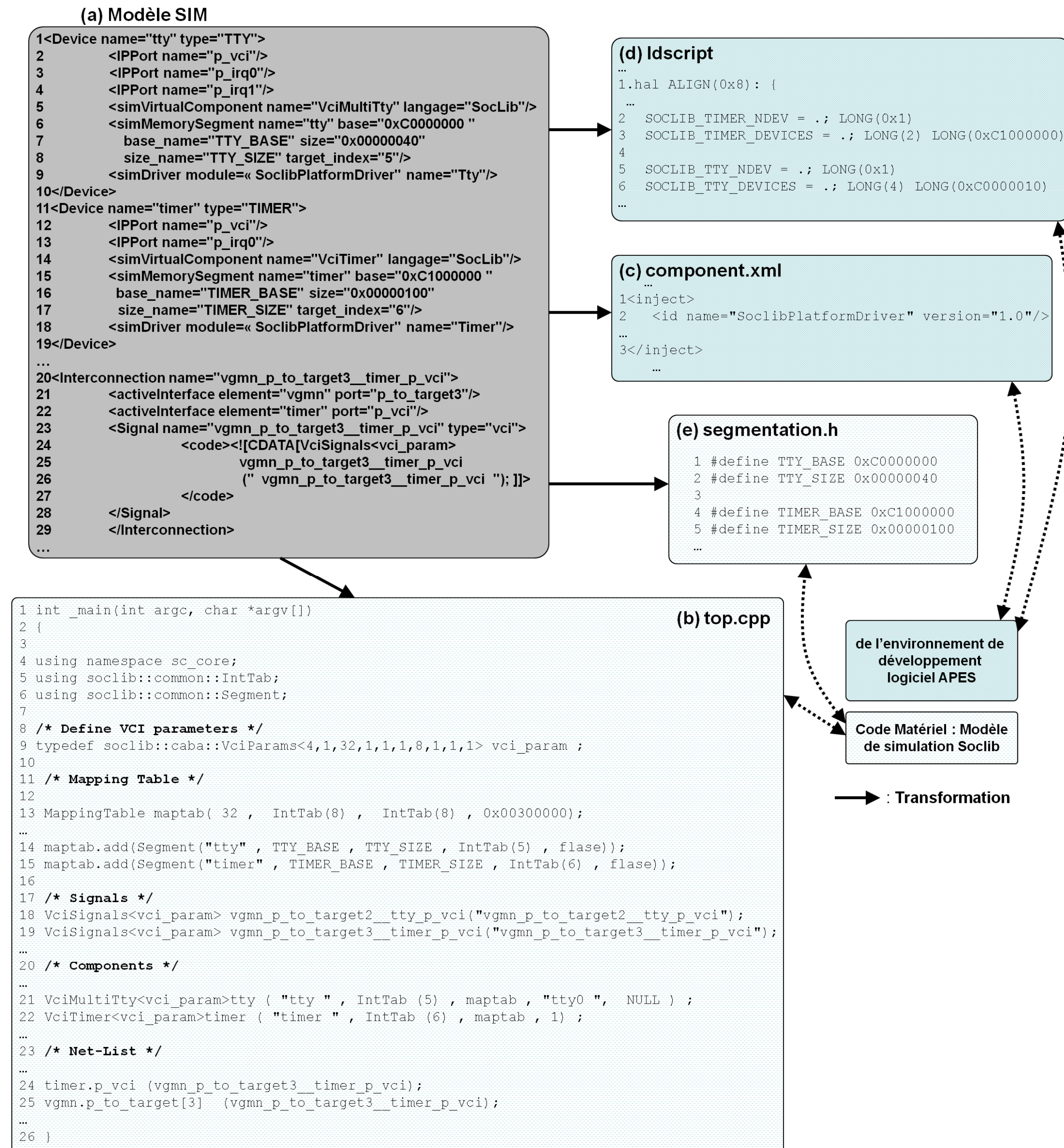


Figure 7.7. génération des configurations de l'environnement APES et du modèle « SoCLib » de simulation à partir du modèle intermédiaire SIM

7.9. Conclusion

Dans ce chapitre, nous avons montré la possibilité de généraliser le flot de génération de code de configuration afin de supporter la génération de code pour les architectures multiprocesseurs. Ceci est possible à travers l'intégration des environnements de développement logiciel et matériel dans la partie « Back-End ». Cette intégration passe à travers la génération des entrées, spécifiques aux environnements de développement, à partir du modèle intermédiaire spécifique « SIM ». Nous avons réussi à rassembler des informations supplémentaires, et spécifiques aux architectures multiprocesseurs, dans l'élément IP-XACT « designConfiguration » étendu. La partie « Front-End », quant à elle, a été exploitée de la même manière dans le déploiement des applications flots de données (3GPP-LTE, MJPEG) sur les plateformes Magali et multiprocesseurs.

Chapitre 8. Conclusion et Perspectives

8.1. Conclusion

Les flots classiques de génération de code permettent le déploiement d'applications orientées flot de données sur des architectures multiprocesseurs. Nous avons montré que ces flots ne peuvent pas être utilisés pour d'autres types d'architectures, comme celles à base d'IPs interconnectés par un réseau sur puce. Dans ce type d'architecture, le code est constitué par un ensemble de configurations exécutées par les interfaces réseau configurables et par les IPs.

Dans cette thèse, nous avons essayé de répondre aux besoins de programmation de ce nouveau type d'architecture, et de traiter les problèmes liés à la génération automatique du code de configuration.

En premier lieu, nous avons structuré ce type de code. Nous avons défini une organisation en couches en spécifiant les fonctionnalités fournies par chacune. Cela permet de faciliter le développement du code de configuration et sa génération automatique.

Pour palier le problème de manque de flot de génération de code de configuration, nous avons proposé une approche basée sur des modèles de haut niveau pour la description des applications orientées flots de données (APDL), des architectures configurables à base d'IPs (ARDL) et du partitionnement des ressources. Nous avons introduit le contrat de partitionnement qui précède le partitionnement des ressources proprement dit. Il s'agit de décrire les règles de partitionnement relatives aux caractéristiques des applications et des plateformes cibles. Ces règles sont prises en considération et validées durant la phase de partitionnement. Le contrat de partitionnement est une nouveauté qui permet de représenter des associations de ressources non modélisables par les modèles de partitionnement classiques.

Nous avons traité le problème de spécificité des flots et des modèles de haut niveau, par rapport aux types d'applications et d'architectures, en séparant notre flot en deux parties distinctes. La première phase « FRONT-END » est indépendante des détails de la plateforme cible alors que la deuxième phase « BACK-END » en est dépendante. Chaque phase est identifiée par un modèle intermédiaire. Le « GIM » rassemble les composants applicatifs et d'architectures provenant des modèles APDL et ARDL, en appliquant le partitionnement des ressources défini dans le modèle de partitionnement. Le modèle intermédiaire « SIM » est un raffinement du modèle « GIM ». Il comporte des informations précises concernant la plateforme cible et représente un modèle proche du code final à générer.

Le flot proposé peut être vu comme en mise en œuvre d'une approche MDE. La grammaire des modèles de représentations de l'application, de l'architecture et du partitionnement est décrite avec des méta-modèles (des schémas XSD, des modèles UML, etc.). Les deux modèles intermédiaires « GIM » et « SIM » correspondent alors respectivement aux modèles « PIM » et « PSM ». Enfin, les outils de génération des modèles

intermédiaires et du code final peuvent être considérés comme des outils de transformation de modèles.

Le passage par ces deux modèles intermédiaires rend le flot flexible et facilite le développement des outils de génération de code. Ces derniers sont plus couteux et difficiles à développer si on part du modèle « GIM » pour la génération finale du code de configuration, sans passer par un second modèle intermédiaire.

Afin d'accélérer et automatiser la génération des configurations, nous avons spécifié et implanté notre flot en se basant sur le standard IEEE 1685 (IP-XACT). Ce dernier est adapté à la représentation et l'accès aux informations (API TGI) des architectures matérielles et en particulier des composants IPs. Il fournit également un mécanisme d'automatisation des flots de génération de code à travers des mécanismes d'intégration et de séquencement de divers outils. Dans un premier temps, nous avons proposé plusieurs extensions au standard IEEE 1685 pour permettre la représentation en ARDL des composants IPs matériels et des chemins de communication physiques.

Nous avons ensuite intégré le flot, dans l'environnement industriel Magillem. Nous avons utilisé les outils MIP et MPA de l'environnement Magillem pour « packager » et assembler, en ARDL, les IPs et les plateformes cibles. Nous avons intégré également le processus d'automatisation des outils du flot dans le module MGS de l'outil Magillem.

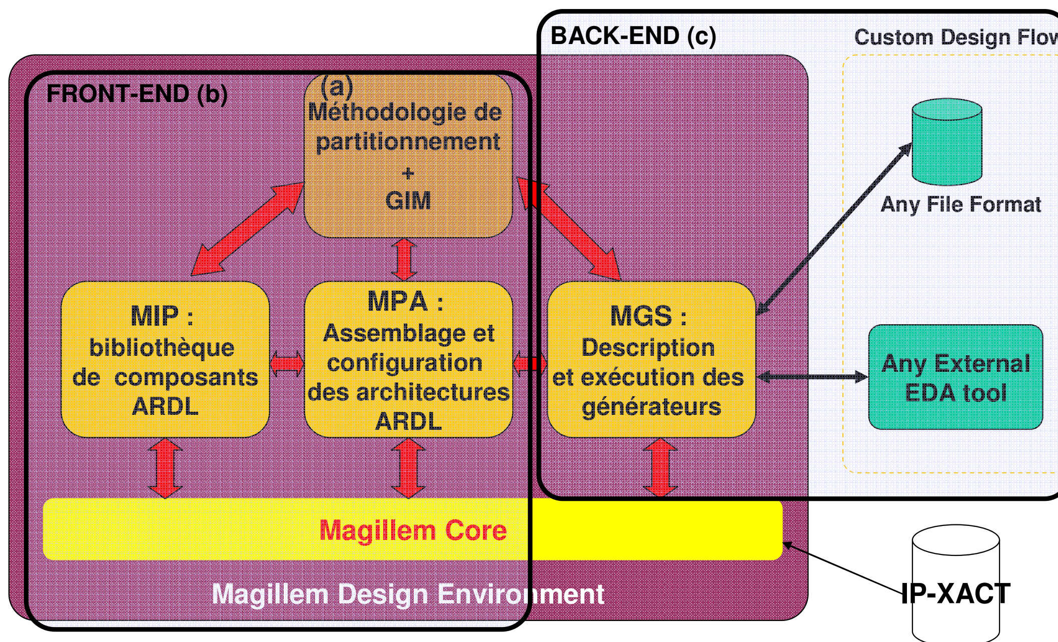


Figure 8.1. Intégration du flot de génération de code de configuration dans l'environnement industriel Magillem

La partie « FRONT-END » (Figure. 8.1.b) du flot est réalisable par les outils MIP et MAP de l'environnement Magillem, en plus du plugin Eclispe que nous avons développé et intégré à l'environnement Magillem. Ce plugin permet la modélisation de l'application en APDL ainsi que la modélisation et la réalisation de la méthodologie de partitionnement pour l'obtention du modèle intermédiaire « GIM ». La partie « BACK-END » (Figure. 8.1.c) du

flot est réalisable en ordonnançant les outils de génération de code de configuration avec l'outil MGS de l'environnement Magillem.

Le flot a été validé pour le déploiement de l'application 3GPP-LTE sur la plateforme Magali. Le développement manuel des fichiers de configuration, peut prendre plusieurs semaines, voire quelques mois. L'utilisation de notre flot, intégré dans l'environnement Magillem, permet d'améliorer ce temps qui devient de l'ordre d'une à deux semaines.

8.2. Perspectives

Le travail présenté dans cette thèse peut être utilisé, pour déployer des applications sur des architectures de nouvelles générations. Il s'agit de plateformes contenant à la fois des IPs, des interfaces réseau configurables et des processeurs exécutants des tâches applicatives. Dans des travaux futurs, nous approfondirons le flot commun de génération de code proposé dans le chapitre 7 pour supporter plusieurs environnements de développement de code. Le flot commun servira pour la génération, à la fois du code de configuration et du code logiciel embarqué pour les plateformes hétérogènes.

Dans le cas de la plateforme Magali, notre flot a été validé pour générer une grande partie des configurations et des microprogrammes pour déployer des applications de télécommunication. Une partie des configurations n'a pas encore été générée. Elle est constituée des fichiers de configuration « ip.sno » indispensables pour chaque IP et du fichier « cpu.sno » qui décrit le comportement global de l'application. Un travail d'extension du modèle intermédiaire spécifique « SIM » est indispensable pour la génération de ces fichiers de configuration. Cette extension doit rendre possible la description des informations telles que les adresses des configurations de communication et de contexte en plus des adresses d'interruption de chaque IP.

A ce jour, le partitionnement des ressources est défini d'une manière manuelle. Dans ce travail, nous nous sommes intéressés uniquement à l'automatisation de la génération du code de configuration à partir d'un partitionnement déjà défini. Nous envisageons d'apporter des heuristiques de placement des ressources qui peuvent aider le concepteur à spécifier par exemple le nombre de contrôleurs de communication à activer (« ICC » et « OCC »), le besoin d'utilisation ou non des composants « SME », etc.

Un prochain travail est envisageable pour fusionner l'outil « ComC », qui permet la génération de code de configuration sous format binaire, avec le flot que nous avons proposé. Il s'agit d'utiliser les modèles de haut niveau que nous avons définis dans cette thèse comme des modèles d'entrée pour l'outil « ComC ». Ainsi nous voulons disposer d'un unique environnement de génération de code de configuration qui servira pour la simulation et pour l'exécution réelle en partant des mêmes descriptions de haut niveau de l'application, de l'architecture et du partitionnement des ressources.

Le flot que nous avons développé dans un environnement IP-XACT permet le déploiement des applications dont le flux des données est fixe et connu dès le départ. Il se trouve que ces applications pourraient disposer de flux de données dynamiques. Une évolution des mécanismes de génération de code est envisageable pour traiter de tels flux.

Des travaux futurs sont également envisageables pour déployer plusieurs applications à la fois sur la même plateforme cible.

Références bibliographiques

[Adr03] Adrijean Adriahantenaina, Hervé Charlery, Alain Greiner, Laurent Mortiez, Cesar A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network", DATE 2003, pp. 20070-20073.

[Altova] XmlSpy, site web : http://www.altova.com/

[And08] Charle André, Frédéric Mallet, Aamir Mehmood Khan, Robert de Simeone, "Modeling SPIRIT IP-XACT with UML MARTE", in DATE Workshop on MARTE, DATE 2008, pp. 35-40.

[Apes] Xavier Guérin, Le projet APES : http://tima-sls.imag.fr/www/research-projects/apes

[Ati08] David Atienza, Federico Angiolini, Srinivasan Murali, Antonio Pullini , Luca Benini, Giovanni De Micheli, "Network-On-Chip Design and Synthesis Outlook", Integration The VLSI journal, vol. 41, num. 2, Feb 2008, pp. 340-359

[Ben01] Luca Benini, Giovanni D. Michelli, "Powering network-on-chips", in the 14th International Symposium onSystem Synthesis (ISSS), 2001, pp. 33-38.

[Ben05] Luca Benini, Davide Bertozzi, "Network on chip and design methods", Computers and Digital Techniques, IEEE Proceedings, Volume 152, Issue 2 , Mars 2005, pp. 261- 272.

[Ber04] Davide Bertozzi, Luca Benini, "Xpipes: A Network-On-Chip Architecture for Gigascale Systems-on-Chips", IEEE Circuits and Systems Magazine, v.4(2), 2004, pp. 18-31.

[Ber05] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, Giovanni De Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip", IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 2, Feb 2005, pp. 113–129.

[Bil95] Greet Bilsen, Marc Engels, Rudy Lauwereins, J. A. Peperstraete, "Cyclo-static data flow". In IEEE Int. Conf. ASSP, Detroit, Michigan, May 1995, pp. 3255–3258.

[Bje06] Tobias Bjerregaard et Shankar Mahadevan, "A Survey of Research and Practices of Network-on-Chip", ACM Computing Surveys (CSUR), 2006, vol. 38.

[Bou03] Pierre Boulet, Jean-Luc Dekeyser, Cédric Dumoulin, Philipe Marquet, "MDA for SoC Design, Intensive Signal Processing Experiment", FDL 2003, pp. 309-317.

[Bou07] Pierre Boulet, Philippe Marquet, Éric Piel, Julien Taillard, "Repetitive Allocation Modeling with MARTE", in Forum on specification and design languages (FDL'07), Barcelona, Spain, September 2007. pp. 280-285.

[But97] David R. Butenhof, "Programming with POSIX Threads", Addison Wesley, May,1997, ISBN 0201633922.

[Ces01] Wander O. Cesário, Gabriela Nicolescu, Lovic Gauthier, Damien Lyonnard, Ahmed Amine Jerraya, "Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design", IEEE International Workshop on Rapid System Prototyping 2001, pp. 110-115.

[Cha06] Herve Charlery, Alain Greiner, "A SystemC Test Environment For Spin Network", MIXDES 2006, pp. 449-453.

[Chu08] Alexandre Chureau, Frédéric Pétrot, "An Intermediate Format for Automatic Generation of MPSoC Virtual Prototypes", SAMOS VIII 2008 , Greece, pp.165-172.

[Chw08] Antoine Chwartz, "Développement d'un outil de mapping d'applications pour les plateformes utilisant un réseau sur puce de type FAUST", mémoire de DRT, 2008.

[Cle05] Fabien Clermidy, Didier Varreau, Didier Lattard, "A NoC-based communication framework for seamless IP integration in complex systems", Proceedings of Design and Reuse IP-SOC'2005, pp. 279-283.

[Cle09] Fabien Clermidy, Romain Lemaire, Xavier Popon, Dimitri Knetas, Yvain Thonnart, "An open and reconfigurable platform for 4G Telecommunication: concepts and application", 12th Euromicro Conference on Digital System Design (DSD'2009), 27-29 août 2009, Patras, pp.449-456.

[CLTV09] Fabien Clermidy, Romain Lemaire, Yvain Thonnart, Pascal Vivet, "A Communication and Configuration Controller for NoC based Reconfigurable Data Flow Architecture", 3rd IEEE International Symposium on Networks-on-Chip (NoCs 2009), pp. 153-162.

[Cop08] Marcello Coppola, Miltos D. Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia and Lorenzo Pieralisi, "Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC." CRC Press, Inc. 2008.

[Cuc05] Arnaud Cuccuru, Jean-Luc Dekeyser, Philippe Marquet, Pierre Boulet, "Towards UML 2 Extensions for Compact Modeling of Regular Complex Topologies", MoDELS 2005, pp. 445-459.

[Daf07] Rachid Dafali, Jean-Philippe Diguet, Samuel Evain, Yvan Eustache, Emmanuel Juin, "µSpider CAD Tool: case study of NoC IP generation for FPGA", in Proc. of Workshop on DASIP'07, Grenoble, France, Nov 27-29 2007.

[Dem95] Alain Demeure, Anne Lafage, Emmanuel Boutillon, Didier Rozzonelli, Jean-Claude Dufourd, and Jean-Luis Marro, "Array-OL : Proposition d'un formalisme tableau pour le traitement de signal multi-dimensionnel". 15ème colloque Gretsi, Juan-Les-Pins, France, 1995, pp. 1029-1032.

[Dom09] Rainer Dömer, Andreas Gerstlauer, Wolfgang Müller, "Introduction to hardware-dependent software design hardware-dependent software for multi- and many-core embedded systems". Proceedings of the 2009 Conference on Asia and South Pacific Design Automation (ASP-DAC '09), Piscataway, NJ, USA, pp. 290-292.

[Dsx] Nicolas Pouillon, Alain Greiner. Projet DSX : https://www-asim.lip6.fr/trac/dsx , 2006-2010.

[Duolog] Duolog, site web : http://www.duolog.com.

[Dur05] Yves Durand, Christian Bernard, Didier Lattard, "FAUST: On-Chip Distributed SoC Architecture for a 4G Baseband Modem Chipset." Proceeding of Design and Reuse IP-SOC' 2005, pp. 51–55.

[Eva06] Samuel Evain, Jean-Philippe Diguet and Dominique Houzet, "µSpider NoC Road Map", DATE 06 Workshops, Future Interconnects and Networks on Chip Workshops, March 10, 2006.

[Fau06] Etienne Faure, Alain Greiner, Daniela Genius, "A generic hardware/software communication mechanism for Multi-Processor System on Chip, targeting telecommunication applications", In Proceedings of the 2006 conference on Reconfigurable Communication-centric SoCs, ReCoSoC 2006, pp. 237-242.

[Fer04] Moraes Fernando, Calazans Ney, Mello Aline, Möller Leandro, Ost Luciano, "Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip", Integration the VLSI Journal, 38(1), Oct. 2004, pp. 69-93.

[Gam10] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Anne Etien, Rabie Ben Atitallah, Philippe Marquet, Jean-Luc Dekeyser, "A Model Driven Design Framework for Massively Parallel Embedded Systems". ACM Transactions in Embedded Computing Systems (TECS), 2010.

[Gar09] Alexandre Chagoya-Garzon, Xavier Guerin, Frédéric Rousseau, Frédéric Pétrot, Davide Rossetti, Alessandro Lonardo, Piero Vicini, Pier Stanislao Paolucci, " Synthesis of Communication Mechanisms for Multi-tile Systems Based on Heterogeneous Multi-processor System-On-Chips", IEEE International Workshop on Rapid System Prototyping 2009, pp. 48-54.

[GDGPRR05] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago G. Pestana, Andrei Radulescu, Edwin Rijpkema, "A Design Flow for application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", DATE 2005, pp. 1182-1187.

[Goo05] Kees Goossens, John Dielissen, Andrei Radulescu, "Æthereal network on chip: concepts, architectures, and implementations", Design & Test of Computers, IEEE Sept.-Oct. 2005 Volume: 22, Issue: 5, pp. 414- 421.

[Gue00] Pierre Guerrier, Alain Greiner, "Generic Architecture for On-chip Packet-switched Interconnections", DATE 2000, Paris, France, pp. 250-256.

[Gue07] Xavier Guerin, Katalin Popovici, Wassim Youssef, Frederic Rousseau, Ahmed A. Jerraya, "Flexible Application Software Generation for Heterogeneous Multi-Processor System-on-Chip" , 31st Annual International Computer Software and Applications Conference compsac 2007 , vol. 1, pp.279-286.

[Gue09] Xavier Guerin, Frédéric Pétrot, "A System Framework for the Design of Embedded Software Targeting Heterogeneous Multi-core SoCs", ASAP 2009, pp. 153-160.

[Han07] Andreas Hansson, Kees Goossens, "Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases", IEEE International Symposium on Networks-on-Chip (NoCs), May 2007, pp. 233-242.

[Han09] Sang-Il Han, Soo-Ik Chae, Lisane B. de Brisolara, Luigi Carro, Katalin Popovici, Xavier Guerin, Ahmed Amine Jerraya, Kai Huang, Lei Li, Xiaolang Yan, "Simulink®-based heterogeneous multiprocessor SoC

design flow for mixed hardware/software refinement and simulation". Integration (2009), volume 42, no 2, pp. 227-245.

[Hospi] Livrables D2.4 et D3.1 du projet ANR HOSPI (HOmegeneous Specification for Platform Integration), "Second Application Mapping Tool, Computation and Communications", site web du projet : http://tima.imag.fr/hospi/.

[Hua07] Kai Huang, Sang-Il Han, Katalin Popovici, Lisane B. de Brisolara, Xavier Guerin, Lei Li, Xiaolang Yan, Soo-Ik Chae, Luigi Carro, Ahmed Amine Jerraya, "Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264", DAC 2007, pp. 39-42.

[Huy08] Dinh Thuy Phan Huy, Rodolphe Legouable, Dimitri Ktenas, Loïc Brunel, Mohamad Assaad, "Downlink B3G MIMO OFDMA Link and System Level Performance", VTC Spring 2008, pp. 1975-1979.

[Ip-xact] SPIRIT consortium Mars 2009, IP-XACT Draft/D3: A specification for XML meta-data and tool interfaces -User Guide v1.5.

[Jal04] Antoine Jalabert, Srinivasan Murali, Luca Benini, Giovanni De Micheli, "xpipesCompiler: a tool for instantiating application specific networks on chip", DATE 2004, pp. 884-889.

[Jer06] Ahmed A. Jerraya, Aimen Bouchhima, Frédéric Pétrot, "Programming models and HW/SW interfaces abstraction for multi-processor soc". Proceedings of the 43rd annual conference on Design automation (DAC '06), New York, USA, pp 280-285.

[Lat08] Didier Lattard, Edith Beigne, Fabien Clermidy, Yves Durand, Romain Lemaire, Pascal Vivet, Friedbert Berens, "A Reconfigurable Baseband Platform Based on an Asynchornous Network-on-Chip", IEEE Journal of Solid-State Circuits, Vol 43, Issue 1, Jan. 2008, pp. 223-235.

[Lee87] Edward A. Lee, David G. Messerschmitt, "Synchronous Data Flow", Proceedings of the IEEE, vol. 75, no. 9, September, 1987. pp. 1235-1245.

[Lee95] Edward A. Lee and Thomas M. Parks, "Dataflow Process Networks", in Proceedings of the IEEE, volume. 83, no. 5, pp. 773-799, 1995

[Magillem] Magillem Design Service, site web: www.magillem.com.

[Marte] MARTE specification : http://www.omgmarte.org/.

[Mur04] Srinivasan Murali, Giovanni De Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs", DAC 2004, pp. 914-919.

[Mur05] Srinivasan Murali, Giovanni De Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation", DATE 2005, pp. 1176-1181.

[Nauet] Nauet, site web : http://www.mataitech.com.

[Nia04] Pierre Niang, Thierry Grandpierre, Mohamed Akil, Yves Sorel,"AAA and SynDEx-Ic: A Methodology and a Software Framework for the Implementation of Real-Time Applications onto Reconfigurable Circuits", in Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, pp. 1119-1123.

[Nik06] Hristo Nikolov, Todor Stefanov, Ed F. Deprettere, "Multi-processor system design with ESPAM", CODES+ISSS 2006, pp. 211-216

[Nik08] Hristo Nikolov, Mark Thompson, Todor Stefanov, Andy D. Pimentel, Simon

Polstra, R. Bose, Claudiu Zissulescu, Ed F. Deprettere, " Daedalus: toward composable multimedia MP-SoC design", Proceedings of the 45th annual Design Automation Conference, DAC 2008, pp. 574-579.

[Pie08] Éric Piel, Rabie Ben Attitalah, Philippe Marquet, Samy Meftali, Smaïl Niar, Anne Etien, Jean-Luc Dekeyser, Pierre Boulet, "Gaspard2: from MARTE to SystemC Simulation", in Proceeedings of the DATE'08 workshop on Modeling and Analyzis of Real-Time and Embedded Systems with the MARTE UML profile, March 2008.

[plugin] Plugin Eclipse pour l'édition et la vérification des modèles IP-XACT, site web : http://download.eclipse.org/dsdp/dd/downloads.

[Pop08] Katalin Popovici, Xavier Guerin, Frédéric Rousseau, Pier Stanislao Paolucci, Ahmed Amine Jerraya, "Platform-based software design flow for heterogeneous MPSoC". ACM Trans. Embedded Computing Systems, volume 7, issue 4, article 39, 2008.

[Pop09] Katalin Popovici, Frederic Rousseau, Ahmed Amine Jerraya, Marilyn Wolf, "Embedded Software Design and Programming of Multiprocessor System-on-chip: Simulink and System C Case Studies". New York, NY, USA, Springer, ISBN: 1441955666

[Rad04] Andrei Radulescu, John Dielissen, Kees Goossens, Edwin Rijpkema, Paul Wielage, "An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration", DATE 2004, pp. 878-883.

[Rau06] Mickaël Raulet, Fabrice Urban, Jean-François Nezan, Christophe Moy, Olivier Deforges, Yves Sorel, "Rapid Prototyping for Heterogeneous Multicomponent Systems: An MPEG-4 Stream over a UMTS Communication Link", EURASIP journal on applied signal processing, 2006 , no 14, pp 1-13.

[Rev08] Sebastien Revol, Safouan Taha, François Terrier, Alain Clouard, Sébastien Gérard, Ansgar Radermacher, Jean-Luc Dekeyser, "Unifying HW Analysis and SoC Design Flows by Bridging Two Key Standards: UML and IP-XACT", DIPES 2008, pp. 69-78.

[Scarlet] Scarlet, site web : http://www.scarletcode.co.uk.

[Soclib] SoCLib Consortium, "Projet SoCLib : Plate-forme de modélisation et de simulation de systèmes integrés sur puce", Technical report, CNRS, 2003. http ://www.soclib.fr.

[Spo07] Thomas Sporer, Michael Beckinger, Andreas Franck, Iuliana Bacivarov, Wolfgang Haid, Kai Huang, Lothar Thiele, Pier S. Paolucci, Piergiovanni Bazzana, Piero Vicini, Jianjiang Ceng, Stefan Kraemer, Rainer Leupers, "SHAPES - a Scalable Parallel HW/SW Architecture Applied to Wave Field Synthesis", in Proceedings of the International Conference on Audio Engineering Society (AES 2007), Hillerod, Denmark, pp. 175-187.

[Stu06] Sander Stuijk, Mark Geilen, Twan Basten, "SDF3: SDF For Free",ACSD 2006, Turku, Finland, June 2006, pp. 276-278.

[Systemc] SystemC, site web : http://www.systemc.org/home/.

[Thi07] Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, Kai Huang, "Mapping Applications to Tiled Multiprocessor Embedded Systems", Application of Concurrency to System Design, ACSD 2007, pp. 29-40.

[Tho07] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, Ed F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs", CODES+ISSS 2007, pp. 9-14.

[Ver07] Sven Verdoolaege, Hristo Nikolov, Todor Stefanov, "pn: a Tool for Improved Derivation of Process Networks", EURASIP Journal on Embedded Systems, 2007, 13 pages.

[Zim80] Hubert Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", IEEE Transactions on Communications, vol. 28, no. 4, April 1980, pp. 425 - 432.

Publications

1- Amin El Mrabti, Frédéric Pétrot, Aimen Bouchhima, « Extending IP-XACT to support an MDE based approach for SoC design », DATE 2009, pp. 586-589.

2- Amin El Mrabti, Hamed Sheibanyrad, Frédéric Rousseau, Frédéric Pétrot, Romain Lemaire, Jérome Martin, « Abstract Description of System Application and Hardware Architecture for Hardware/Software Code Generation », DSD 2009, pp. 567-574.

3- Amin El Mrabti, Frédéric Rousseau, Frédéric Pétrot, Jérome Martin, Romain Lemaire, Emmanuel Vaumorin, « Design Environment for the Support of Configurable Network Interfaces in NoC based Platforms », SAMOS 2010, pp. 63-70

Annexes

Annexe A : Extension d'IP-XACT pour le modèle de représentation ARDL

```
<xs:complexType name="IPElement">
  <xs:sequence>
     <xs:choice>
       <xs:element name="EU" type="ardl:EU"/>
       <xs:element name="HWIP" type="ardl:HWIP"/>
       <xs:element name="device" type="ardl:device"/>
       <xs:element name="storageDevice" type="ardl:storageDevice"/>
       <xs:element name="router">
        <xs:complexType mixed="false">
            <xs:complexContent mixed="false">
                  <xs:extension base="ardl:router"/>
            </xs:complexContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="networkInterface"
type="ardl:networkInterface"/>
      </xs:choice>
      <xs:element name="HALAPI" minOccurs="0">
         <xs:complexType>
            <xs:sequence>
              <xs:element name="HALPrimitive" type="ardl:HALPrimitive"
maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="metaTag" type="ardl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
</xs:complexType>
```

Figure A.1. Elément « IPElement » dans le schéma «ARDLComponentVendorExtensions.xsd »

```
<xs:complexType name="HALPrimitive">
  <xs:sequence>
    <xs:choice>
      <xs:element name="parameter" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
      <xs:element name="value" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:string">
      <xs:attribute name="occurence" type="xs:string"
use="optional"/> <xs:attribute name="argumentType" type="xs:string"
use="optional"/>
      <xs:attribute name="argumentValue" type="xs:string"
use="optional"/>
                  </xs:extension>
               </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="metaTag" type="ardl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
         </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="service" type="ardl:serviceElement"/>
   </xs:choice>
    <xs:element name="metaTag" type="ardl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
 </xs:sequence>
<xs:attribute name="primitiveName" type="xs:string" use="optional"/>
<xs:attribute name="primitiveOccurence" type="xs:string"
use="optional"/>
</xs:complexType>
```

Figure A.2. Elément « HALPrimitive » dans le schéma «ARDLComponentVendorExtensions.xsd »

```
<xs:complexType name="ardlComponentInstanceElement">
  <xs:attribute name="position" type="xs:string" use="optional"/>
  <xs:attribute name="directionType" use="optional">
     <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Ressource"/>
            <xs:enumeration value="SOUTH"/>
            <xs:enumeration value="NORTH"/>
            <xs:enumeration value="EAST"/>
            <xs:enumeration value="WEST"/>
        </xs:restriction>
     </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

Figure A.3. Elément « ComponentInstanceElement » dans le schéma «ComponentInstanceVendorExtensions.xsd »

```
<xs:complexType name="comLinkInterconnection">
   <xs:sequence>
     <xs:element name="name">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string"/>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="sourceInterface" type="xs:string"/>
    <xs:element name="destinationInterface" type="xs:string"/>
    <xs:element name="metaTag" type="ardl:metaTag" minOccurs="0"/>
    <xs:element name="accessType" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Figure A.4. Elément « ComLinkInterconnection » dans le schéma «DesignVendorExtensions.xsd »

```
<xs:element name="simComponentConfiguration" minOccurs="0"
maxOccurs="unbounded">
     <xs:complexType>
          <xs:sequence>
            <xs:element name="gimInstanceRef" type="xs:string"/>
            <xs:element name="requires" minOccurs="0"
maxOccurs="unbounded">
               <xs:complexType>
                 <xs:sequence>
                    <xs:element name="DriverRef" type="xs:string"
minOccurs="0"/>
                    <xs:element name="VirtualComponent">
                       <xs:complexType>
<xs:attribute name="componentName" type="xs:string" use="required"/>
<xs:attribute name="virtualcomponentName" type="xs:string"
use="required"/>
<xs:attribute name="langage" type="xs:string" use="required"/>
                       </xs:complexType>
                   </xs:element>
                 </xs:sequence>
               </xs:complexType>
           </xs:element>
<xs:element name="configuration" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                 <xs:sequence>
                      <xs:element name="name" minOccurs="0"/>
                      <xs:element name="parameter"
maxOccurs="unbounded"/>
                 </xs:sequence>
                </xs:complexType>
           </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure A.5. Element « simComponentConfiguration » dans le schéma « designConfigurationVendorExtensions.xsd »

```
<xs:element name="simSegmentConfiguration" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="gimInstanceRef" type="xs:string"/>
            <xs:element name="segment" minOccurs="0"
maxOccurs="unbounded">
               <xs:complexType>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="base" type="xs:string" use="required"/>
<xs:attribute name="size" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
            <xs:element name="requires" minOccurs="0"
maxOccurs="unbounded">
               <xs:complexType>
<xs:attribute name="componentRef" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure A.6. Element « simSegmentConfiguration » dans le schéma « designConfigurationVendorExtensions.xsd »

```
<xs:element name="simComLinkConfiguration" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="gimComLinkRef" type="xs:string"/>
                  <xs:element name="DriverRef" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

Figure A.7. Element « simComLinkConfiguration » dans le schéma « designConfigurationVendorExtensions.xsd »

Annexe B : Modèle de représentation APDL

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="/infrastructure/apdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:apdl="/infrastructure/apdl">
 <xsd:include schemaLocation="apdl_library.xsd"/>
 <xsd:element name="Application">
     <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="variables">
            <xsd:complexType>
               <xsd:sequence>
<xsd:element name="variable" type="apdl:Variable" minOccurs="0"
maxOccurs="unbounded"/>
               </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="tasks">
            <xsd:complexType>
               <xsd:sequence>
<xsd:element name="task" type="apdl:Task"
maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="channels">
             <xsd:complexType>
               <xsd:sequence>
<xsd:element name="channel" type="apdl:Channel"
maxOccurs="unbounded"/>
               </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="interconnections">
             <xsd:complexType>
                <xsd:sequence>
<xsd:element name="interconnection" type="apdl:Interconnection"
maxOccurs="unbounded"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="name" type="xsd:string"
use="required"/>
     </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure B.1. Schéma « apdl.xsd »

```
<xs:complexType name="Task">
    <xs:sequence>
 <xs:element name="Port" type="apdl:SRSWPort" maxOccurs="unbounded"/>
<xs:element name="metaTag" type="apdl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="TaskAPI" type="apdl:API" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="location" type="xs:string" use="optional"/>
<xs:attribute name="ProcessName" type="xs:string" use="required"/>
      <xs:attribute name="Langage" type="xs:string" use="optional"/>
</xs:complexType>
```

Figure B.2. Elément « Task » dans le schéma « apdl_library.xsd »

```
<xs:complexType name="Channel">
   <xs:sequence>
      <xs:element name="ChannelAccess" minOccurs="2" maxOccurs="2">
        <xs:complexType>
           <xs:sequence>
      <xs:element name="Port" type="apdl:MRMWPort"
maxOccurs="unbounded"/>
      <xs:element name="AccessManagement" type="apdl:Arbiter"
minOccurs="0"/>
      <xs:element name="metaTag" type="apdl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="Type" type="xs:string" use="optional"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="metaTag" type="apdl:metaTag" minOccurs="0"
maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="type" use="optional">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="fifo"/>
            <xs:enumeration value="sharedMemory"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="ChannelName" type="xs:string" use="optional"/>
   <xs:attribute name="largeur" type="xs:int" use="optional"/>
   <xs:attribute name="profondeur" type="xs:int" use="optional"/>
   <xs:attribute name="status" type="xs:int" use="optional"/>
</xs:complexType>
```

Figure B.3. Element « Channel » dans le schéma « apdl_library.xsd »

```
<xs:complexType name="Arbiter">
    <xs:sequence>
        <xs:element name="scenario" maxOccurs="unbounded">
            <xs:complexType>
            <xs:sequence>
              <xs:element name="PortRef" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
<xs:element name="dataDistribution" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
<xs:attribute name="processrefname" type="xs:string" use="required"/>
<xs:attribute name="processoccurence" type="xs:string"
use="required"/>
                        </xs:complexType>
                    </xs:element>
                  </xs:sequence>
<xs:attribute name="portrefname" type="xs:string" use="required"/>
<xs:attribute name="portrefoccurence" type="xs:string"
use="required"/>
                 </xs:complexType>
              </xs:element>
            </xs:sequence>
<xs:attribute name="occurence" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
    </xs:sequence>
<xs:attribute name="Name" type="xs:string" use="required"/>
<xs:attribute name="type" use="required">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Arbiter"/>
                        <xs:enumeration value="Lock"/>
                  </xs:restriction>
            </xs:simpleType>
</xs:attribute>
</xs:complexType>
```

Figure B.4. Elément « Arbiter » dans le schéma « apdl_library.xsd »

```
<xs:complexType name="Interconnection">
      <xs:sequence>
         <xs:element name="name">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:string"/>
               </xs:simpleContent>
      </xs:complexType>
         </xs:element>
<xs:element name="activeInterface" minOccurs="2" maxOccurs="2">
         <xs:complexType>
<xs:attribute name="componentRef" type="xs:string" use="required"/>
<xs:attribute name="busRef" type="xs:string" use="required"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
</xs:complexType>
```

Figure B.5. Elément « Interconnection » dans le schéma « apdl_library.xsd »

Annexe C : Partitionnement des ressources et générateurs IP-XACT

```
<xs:element name="SpecificMappingRules" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MappingConstraint" maxOccurs="unbounded">
        <xs:complexType>
          <xs:choice>
            <xs:sequence>
              <xs:element name="application_ResourceInstance"
type="xs:string"/>
             <xs:element name="hardware_ResourceInstance"
type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
             <xs:element name="MappingParameter" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
         <xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="value" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:sequence>
    <xs:element name="hardware_ResourceInstance" type="xs:string"/>
    <xs:element name="application_ResourceInstance" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="MappingParameter" minOccurs="0"
maxOccurs="unbounded">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>

      <xs:attribute name="value" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:choice>
  <xs:attribute name="mapping_constraint_name" type="xs:string"
use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure C.1. Elément « SpecificMappingRules» dans le schéma «metamapping.xsd »

```
<xs:element name="GeneralMappingRules">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="MappingConstraint" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="hardware_ResourceType" type="xs:string"/>
            <xs:element name="application_ResourceType" type="xs:string"
maxOccurs="unbounded"/>
        </xs:sequence>
      <xs:attribute name="mapping_constraint_name" type="xs:string"
use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure C.2. Elément « GeneralMappingRules» dans le schéma «metamapping.xsd »

```
<xs:element name="TaskBinding" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="ressource">
      <xs:complexType>
        <xs:choice>
          <xs:element name="EURef" type="xs:string"/>
          <xs:element name="HWIPRef" type="xs:string"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="TaskPortBinding" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
        <xs:choice>
          <xs:element name="DeviceRef" type="xs:string"
maxOccurs="unbounded">
          </xs:element>
        </xs:choice>
      <xs:attribute name="TaskPortRef" type="xs:string" use="required"/>
      </xs:complexType>
     </xs:element>
   </xs:sequence>
   <xs:attribute name="TaskRef" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Figure C.3. Elément « TaskBinding» dans le schéma «mapping.xsd

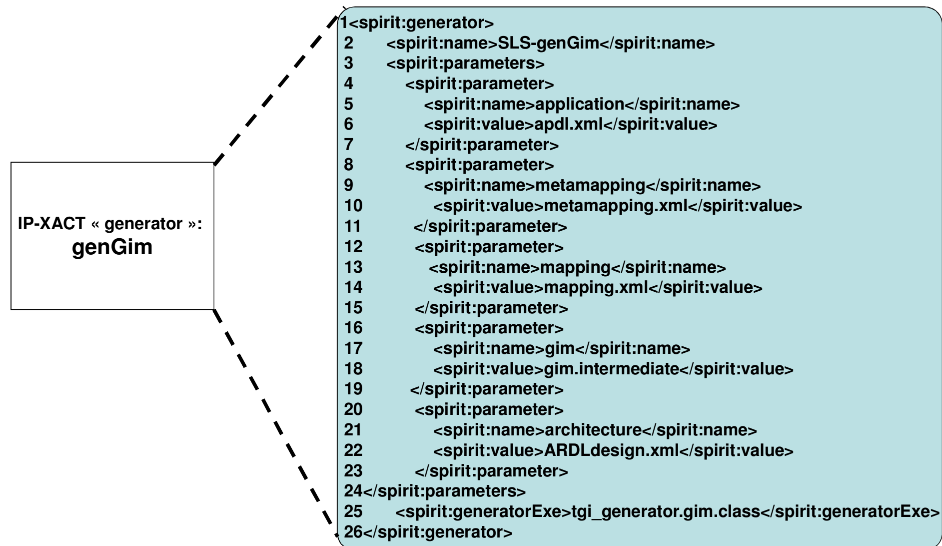


Figure C.4. Description d'un générateur IP-XACT pour la génération du modèle GIM

Résumé La complexité des systèmes sur puce augmente pour supporter les nouvelles applications dans le domaine des télécommunications et du multimédia. La tendance actuelle des nouvelles architectures matérielles converge vers des plateformes multi-cœurs où les unités de calcul (processeurs, DSP, IP) sont interconnectées par un réseau sur puce, dont les interfaces réseau sont configurables. Cette thèse propose un flot de génération de code de configuration pour le déploiement des applications de type flots de données sur les architectures à base d'IPs interconnectés à travers un réseau sur puce configurable. Ce flot est basé sur des modèles de haut niveau de l'application et de l'architecture et du partitionnement des ressources. Le processus de génération de code de configuration passe par plusieurs étapes modélisées par diverses représentations intermédiaires du système. La première ne contient pas d'informations spécifique à la plateforme cible. La deuxième est générée par raffinement de la première représentation intermédiaire. Elle est spécifique à la plateforme cible et elle est exploitée par les outils de génération de code pour produire le code de configuration. Le flot a été développé par la suite dans un environnement basé sur le standard IEEE 1685 (IP-XACT). Le flot proposé a été appliqué pour la génération et la validation du code de configuration en vue de déployer une application 3GPP-LTE de télécommunication sur la plateforme Magali du CEA LETI.

Mots-Clés Interface réseau configurable, Flot de génération de code, Code de configuration, Plateformes multi-cœurs, IEEE 1685(IP-XACT), Modélisation de haut niveau

---

Abstract The complexity of SoC is increasing to support new applications in the telecommunication and multimedia domains. The current trend of the new hardware architectures converges to multi-core platforms which gather several execution units (processors, DSP, IP) interconnected via a network on chip in which the network interfaces can be configured. For such architectures, the classical code generation environments are no longer suitable. This thesis proposes a configuration code generation flow to deploy dataflow applications on IP-based architectures with configurable network on-chip. This flow starts with high level modeling of the application, the architecture and the resources partitioning. The code generation flow goes through several phases which are modeled using various intermediate representations of the system. The first intermediate model does not contain any specific information related to the target platform. The second intermediate representation is generated through the refinement of the first. It is specific to the target platform and is used to produce the configuration code. The code generation flow was developed in an environment based on IEEE 1685 (IP-XACT) standard. The proposed flow has been applied to generate and validate the configuration code to deploy the 3GPP-LTE application on the Magali telecommunication platform.

Keywords Configurable network interface, Code generation flow, Multi-core platforms, Configuration code, IEEE 1685(IP-XACT), High level modeling

ISBN : 978-2-84813-162-7
Laboratoire TIMA – 46 Avenue Félix Viallet, 38000 Grenoble