Vers la reconfiguration dynamique dans les systèmes embarqués: de la modélisation à l'implémentation

Samy Meftali




Mémoire présenté par
Samy Meftali
pour obtenir
l'habilitation à diriger des recherches
en Sciences Mathématiques (spécialité informatique)

Vers la reconfiguration dynamique dans les systèmes embarqués: de la modélisation à l'implémentation

6 juillet 2010

Composition du jury :

Président :
- David Simplot-Ryl : Professeur de l'Université de Lille 1

Rapporteurs :
- Ahmed Jerraya : Directeur de Recherches CEA-LETI, Grenoble
- Michel Robert : Professeur, Université de Montpellier
- Pascal Sainrat : Professeur, Université de Toulouse

Directeur de recherches :
- Jean-Luc Dekeyser : Professeur de l'Université de Lille 1

Examinateurs :
- Dragomir Milojevic : Professeur de l'Université Libre de Bruxelles

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
LIFL - UMR 8022 - Bât. M3 - UFR IEEA - 59655 Villeneuve d'Ascq cedex
Tél. : (33) 03 59 57 78 03 - Fax. : (33) 03 20 43 65 56 - Mél : Samy.Metali@lifl.fr

Remerciements

Je tiens tout particulièrement à exprimer ma profonde gratitude à Jean-Luc Dekeyser, responsable de l'équipe DaRT du LIFL et de l'INRIA Lille – Nord Europe, pour la confiance qu'il m'a témoignée depuis bientôt 8 ans. Son soutien, ses conseils, ses encouragements et sa bonne humeur permanente tout au long de ces années ont joué un rôle déterminant dans l'aboutissement de ce travail. Je lui suis extrêmement reconnaissant d'avoir accepté de superviser mon HDR.

Des remerciements particuliers à Ahmed Jerraya qui a accepté de rapporter sur mes travaux de recherche. C'est lui qui m'a initié au monde de la recherche en encadrant ma thèse de 1999 à 2002. Il suit depuis 11 ans mes travaux, je tiens donc à lui exprimer ma profonde gratitude pour ses précieux conseils, nos discussions diverses à chaque fois que l'on se rencontre, et nos collaborations passées et futures.

Un grand merci aux professeurs Michel Robert et Pascal Sainrat, qui ont répondu favorablement à ma requête pour être rapporteurs de mon HDR. J'ai sollicité leur concours connaissant la qualité scientifique de leurs travaux et les éclairages multiples qu'ils sont susceptibles de donner sur mes propres recherches.

Tous mes remerciements à David Simplot-Ryl pour avoir accepté de présider ce jury et pour l'énergie qu'il déploie pour l'INRIA Lille – Nord Europe. Il donne une dynamique très intéressante à l'institut.

Je tiens à remercier tous les étudiants en doctorat que j'ai encadré pendant ces années, et ceux que j'encadre actuellement. Sans leurs efforts permanents et leur sérieux ce manuscrit n'aurait pas pu voir le jour. Merci à (dans l'ordre chronologique) : Mickael Samyn, Joel Vennin, Rabie Ben Atitallah, Imran Quadri, Chiraz Trabelsi et Sana Cherif. Sans oublier tous les stagiaires de DEA et de master sans qui les prototypes logiciels ne seraient pas ce qu'ils sont.

La bonne ambiance qui règne au LIFL et à l'INRIA Lille – Nord Europe participe aux bonnes conditions qui m'ont permis d'effectuer ce travail. Je remercie donc tous mes collègues avec qui il est agréable de travailler aussi bien en recherche qu'en enseignement. Merci particulièrement à tous mes membres de l'EPI DaRT.

Un grand merci et une pensée particulière à toute ma famille, particulièrement mon épouse Paméla et ma fille Cécilia. Leur amour, soutien et encouragement permanents sont pour beaucoup dans l'aboutissement de mes travaux. Leur présence est ma source d'inspiration.

Résumé

Ce manuscrit résume mes travaux de recherche depuis ma thèse soutenue en septembre 2002. Certains de mes travaux présentés sont achevés à l'heure actuelle, d'autres sont en cours d'avancement ou encore à un stade exploratoire. Tout au long de ces années, mes travaux se sont inscrit dans le contexte de la conception conjointe logicielle/matérielle de SoCs dédiés aux applications de traitement de signal intensif.

La complexité des systèmes ciblant ce domaine d'application ne cesse de *s'accroitre* lors des dernières années. En effet, les besoins grandissants, en terme de puissance de calcul et stockage mémoire des applicatifs du traitement de signal intensif, rendent la conception des Soc les implémentant très fastidieuse et nécessitant un temps et des efforts considérables.

Ainsi, la ligne directrice de mes travaux a toujours été de fournir des méthodes et outils d'aide à la conception de tels SoC, permettant un maximum d'automatisation, une augmentation de la productivité des concepteurs et une réduction des temps de mise sur le marché des systèmes conçus.

Je me suis donc concentré principalement sur trois aspects : la modélisation de haut niveau en fournissant des méta-modèles et profils respectant le standard MARTE ; les plateformes de simulation distribuées, supportant l'interopérabilité entre plusieurs niveaux d'abstraction tout en permettant une bonne estimation de la consommation d'énergie ; et finalement la production d'outils de conception basés sur les transformations automatiques modèle à modèle de l'approche IDM.

Etant convaincu du grand potentiel des FPGAs partiellement et dynamiquement reconfigurables, j'oriente de plus en plus mes travaux pour cibler de telles architectures. Ainsi, mes travaux futurs iront certainement dans le même sens, dans le cadre notamment du projet ANR FAMOUS que je dirige. Ainsi, les grandes orientations de mes recherches concerneront notamment : la modélisation (basée sur MARTE) de la reconfiguration dynamique sous toutes ses facettes (architecture, application, association, déploiement; et partitionnement); la simulation des FPGAs et l'estimation de leur consommation (pour piloter l'exploration d'architectures) ; et enfin l'intégration dans des outils de conception basés sur les standards (tels que MARTE et IDM).

Abstract

This manuscript summarizes my research activities since my thesis defended September 2002. Some of my works presented here are completed; others are in progress or still at an exploratory stage. Throughout these years, my works were in the context of software/hardware co-design of intensive signal processing specific Sacs.

The complexity of systems targeting such application domain continues to expand, in recent years. Indeed, the growing needs in terms of computing power and memory storage of intensive signal processing applications, makes designing SoCs dedicated to them very tedious, requiring considerable time and effort.

Thus, the guideline of my work has always been to provide methods and tools for the design of such Sacs, allowing maximum automation, increasing designer's productivity and reducing time-to-the market of designed systems.

So I concentrated my research effort on three main aspects: high level modeling by providing meta-models and profiles based on the MARTE standard; distributed simulation platforms, supporting interoperability between different abstraction levels while allowing accurate energy consumption estimation; and finally production of design tools based on automatic model to model transformations model and MDE approach.

Being convinced of the great potential of partially and dynamically reconfigurable FPGAs, I focus more and more my work to target such architectures. Thus, my future work will certainly be in the same direction, particularly in the context of the ANR project FAMOUS.

Thus, the main concern of my research will include: modeling (based on MARTE) of dynamic reconfiguration in all its aspects (architecture, application, association, deployment and partitioning), simulation of FPGAs and the estimation of their consumption (to drive architectures exploration), and finally the integration into design tools based on standards (such as MDE and MARTE).

Table des matières

Chapitre 1: Introduction

1.1. Cadre scientifique et économique

Les systèmes sur puce (SoCs) deviennent de plus en plus présents dans nos vies professionnelles et personnelles. On les trouve en grand nombre dans l'avionique, transport, défense, médecine et télécommunications pour n'en citer que ceux là. Les SoCs sont désormais omniprésents et il est difficile, de nos jours, de trouver un domaine où ces systèmes miniaturisés n'ont pas encore laissé leur empreinte.

Ces SoCs modernes étant considérés comme une solution intéressante pour la conception de systèmes embarqués, ils bénéficient aujourd'hui d'un intérêt économique exceptionnel.

Alors, qu'environ 260 millions de processeurs ont été vendus en 2004 pour équiper nos PC de bureau, plus de 14 milliards de processeurs (sous diverses formes : microprocesseur, microcontrôleur, DSP) ont été vendus à la même année pour les systèmes embarqués. Par ailleurs, une augmentation annuelle de 16% du chiffre d'affaire était déjà prévue pour le marché des systèmes embarqués dans les années à venir (Future of Embedded Systems Technology from BCC Research Group).

Aujourd'hui ces prévisions se confirment et le marché mondial des systèmes embarqués représente une valeur de 88,144 millions de dollars, avec un taux de croissance annuel moyen de 14%. De même, le marché du logiciel embarqué représente 3,488 milliards de dollars avec une croissance moyenne de 16%. Cette forte croissance du recours aux systèmes sur puce est certainement due aux nombreux avantages qu'ils offrent comme la réduction de la surface utilisée, la consommation d'énergie et les coûts de fabrication.

Ces systèmes embarqués sur puce sont utilisés dans des domaines d'applications divers et variés. Cependant, ils sont particulièrement présents dans les applications de traitement de signal intensif, où une grande quantité de données est traitée, de manière régulière, au moyen de calculs répétitifs. En plus d'exiger plus de puissance de calcul, ces applications sont souvent soumises à des contraintes temporelles qui doivent être respectées. En outre, afin de se tenir au rythme de l'évolution rapide du matériel, les développeurs de logiciels pour SoCs se doivent d'accroître la capacité de calcul des applications ciblées, pour traiter de grandes quantités de données entrantes dans le système, sur lesquelles les calculs doivent, souvent, être appliqués à des vitesses considérables. L'optimisation de ces fonctionnalités se traduit souvent par la parallelisation des applications et des ressources qui composent le système embarqué. Le parallélisme augmente le nombre de calculs exécutés à un moment tout en limitant (dans certains cas) les niveaux de consommation d'énergie.

La complexité grandissante et les coûts élevés de conception de ces systèmes font aujourd'hui que les concepteurs se tournent de plus en plus vers la reconfiguration matérielle. Il s'agit du fait d'adapter le même matériel (sans repasser par la fonderie) pour exécuter de nouvelles applications (différentes de l'application d'origine). Un SoC reconfigurable offre une grande flexibilité fonctionnelle, tout en maintenant de bonnes performances. Ces systèmes peuvent être reconfigurés un nombre illimité de fois. Ils offrent la possibilité d'ajouter de nouvelles fonctionnalités et apporter des modifications au système après sa fabrication.

La reconfiguration dynamique, qui est un type spécial de reconfiguration, permet la modification du système pendant l'exécution, avec l'introduction du concept de matériel virtuel. Ainsi, les concepteurs peuvent modifier les applications s'exécutant sur ces systèmes (et/ou l'architecture matérielle), en fonction de la qualité de service (QoS) selon des critères

liés à l'environnement extérieur ou la plateforme: tels que la surface utilisée, les niveaux de consommation d'énergie, etc.

Actuellement, les SoCs basés sur le FPGAs (Field Programmable Gate Array) offrent une solution idéale pour mettre en œuvre la reconfiguration dynamique. En outre, les fonctionnalités de l'application peuvent être facilement mises en œuvre comme du matériel reconfigurables sur ces systèmes.

Par rapport à un SoC traditionnel, cet avantage de reconfiguration dynamique peut s'avérer d'une grande efficacité en termes de consommation d'énergie, flexibilité et surface du matériel utilisé. Normalement, ces systèmes intègrent également une sorte de contrôleur de reconfiguration qui gère le processus de reconfiguration entre les régions statiques et dynamiques du système reconfigurable. Ce module est l'un des concepts clés dans le système, et est habituellement associé à certaines sémantiques de contrôle telles que les réseaux de Petri, machines d'état ou automates.

Bien que beaucoup de travaux aient été réalisés pour faciliter la conception de systèmes reconfigurables, de nombreux efforts de recherche ont été principalement motivés par la proposition de nouvelles architectures reconfigurables et optimisations de bas niveau liées fortement aux détails techniques et matériels. Cependant, une attention insuffisante a été accordée à la recherche de moyens de hauts niveaux de conception pour ces systèmes reconfigurables. Ceci est sans doute l'une des raisons principales pour lesquelles la reconfiguration dynamique partielle n'a pas encore pris son envol dans l'industrie des SoCs.

Dans le sillage de l'évolution continue des parties logicielles et matérielles dans les systèmes sur puce (ajout de fonctionnalités telles que la reconfiguration dynamique, nouvelles applications exigeantes en termes de puissance de calcul), la complexité de la conception et le développement de SoCs a dégénéré vers de nouveaux sommets de difficulté, d'une manière exponentielle. Ainsi, les coûts de développement et les délais de mise sur le marché ont augmenté de façon proportionnelle.

Sans l'utilisation d'outils de conception et de méthodologies efficaces, les systèmes sur puce moderne seront de plus en plus difficiles à gérer. En effet, la complexité de ces systèmes rend l'espace d'exploration, qui représente toutes les décisions techniques devant être prises par le concepteur, extrêmement large. De même, la manipulation de ces systèmes à de bas niveaux d'abstraction, comme le niveau transfert de registres (Register Transfer Level - RTL) est souvent très fastidieuse et source de beaucoup d'erreurs.

Il est donc indispensable de compenser la complexité de conception des SoCs en augmentant la productivité des ingénieurs, tout en intégrant les nouvelles technologies d'implémentation comme les FPGA dynamiquement reconfigurables. Ces derniers semblent s'imposer comme solution incontournable, offrant (parmi les autres solutions existantes) un bon compromis flexibilité – performances – adaptabilité (figure 1.1).

Actuellement, nous sommes donc confrontés à la nécessité de concevoir des systèmes sur puce plus efficace à des coûts raisonnables. Diverses méthodologies ont été proposées à cet effet. La conception basée sur les plateformes ou sur la réutilisation de composants sont largement adoptées dans l'industrie de SoCs. Ce sont des approches permettant une grande réutilisation de composants, ayant été développés en interne ou par des tiers. D'autres méthodes utilisent des niveaux d'abstraction élevés, afin de s'affranchir des détails techniques, lors des premières phases de conception.

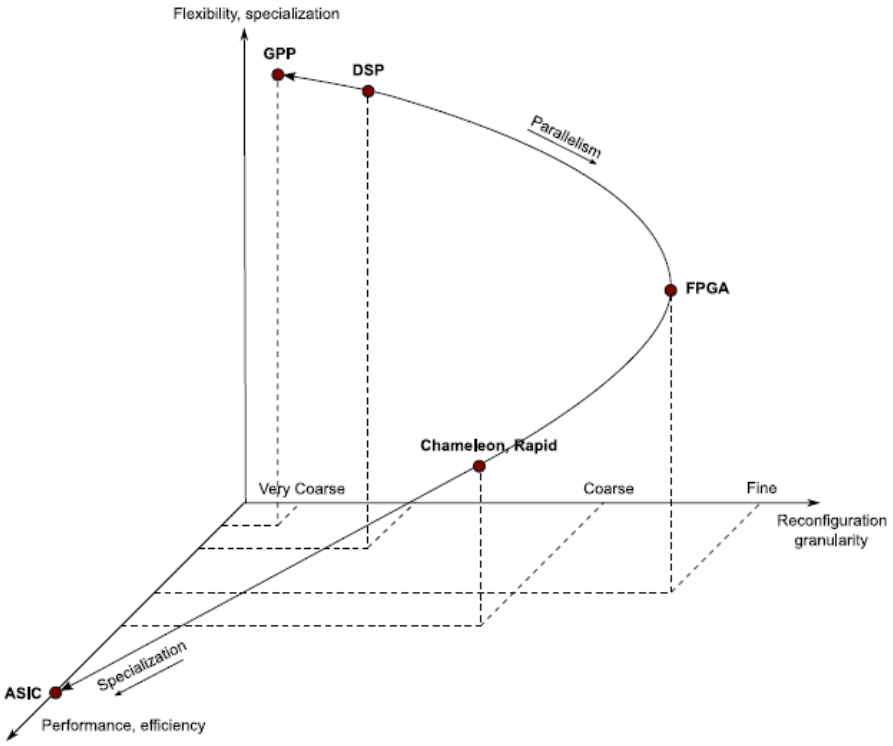


Figure 1.1: Compromis flexibilité, performance, adaptabilité dans les architectures matérielles.

1.2. Défis actuels dans la conception de SoCs

Il semble clair aujourd'hui, que pour que l'industrie des SoCs ait une chance de maintenir sa place dans l'économie, il sera indispensable d'apporter des solutions efficaces à certains défis majeurs. J'en liste trois (sans être exhaustif) ci-dessous.

1.2.1. Temps de mise sur le marché (Time-to-market)

Face aux nouveaux besoins de plus en plus exigeants et afin d'être réactifs aux publications de nouveaux standards et l'apparition de nouvelles technologies d'intégration, les concepteurs de SoCs disposent de très peu de temps pour concevoir et mettre sur le marché leurs systèmes. Cette forte contrainte de temps fait que, les ingénieurs/concepteurs d'aujourd'hui s'orientent de plus en plus vers des systèmes flexibles, avec une forte réutilisation de blocs matériels virtuels (IP).

Cette contrainte de temps est également due, en grande partie, à l'évolution rapide et permanente des technologies d'intégration sur silicium. En effet, une grande partie du temps de conception d'un nouveau système concerne des phases ciblant une technologie d'intégration spécifique (étapes dépendantes de la technologie). La disponibilité de nouvelles technologies ou standards rendent les systèmes, qui les utilisent, obsolètes sur plusieurs points. Ceci ne fait donc que rendre la contrainte de temps de mise sur le marché encore plus forte et importante.

1.2.2. Coût de la conception

L'avenir du semi-conducteur est menacé, aujourd'hui, par son coût de conception extrêmement élevé. Une grande partie de ce coût est liée à la conception du logiciel du SoC.

Ces coûts souvent très lourds pour les constructeurs sont une conséquence directe des temps de modélisation, simulation/validation et intégration.

Sachant qu'une grande partie des systèmes conçus, aujourd'hui, sont abandonnés avant l'étape de commercialisation, les coûts des systèmes arrivant au bout du processus de fabrication augmentent encore plus. Ainsi, à l'époque d'une véritable guerre de prix (des produits finaux) entre fournisseurs, la réduction des coûts de conception semble devenir une condition de survie de tout producteur de SoCs.

Il est donc indispensable de proposer de nouvelles techniques et méthodologies de conception pour réduire ce coût.

1.2.3. Complexité de la conception

L'hétérogénéité des technologies utilisées sur une puce (logique, mémoire, analogique mélangé au digital, systèmes micro/nano électro- mécanique, etc.) demande beaucoup de compétences de la part des ingénieurs pour les faire interagir.

De plus, à la complexité des communications s'ajoutant aux multiples fréquences d'horloge auxquelles ces ressources fonctionnent. Tous ces facteurs influent considérablement sur la complexité de la conception des SoCs.

Ainsi, on parle aujourd'hui de systèmes intégrants plusieurs centaines de millions de transistors. Ceci rend leur manipulation d'une grande difficulté, particulièrement à de bas niveaux d'abstraction.

1.3. Eléments de réponse aux problèmes de conception des Socs

Beaucoup de solutions peuvent être apportées pour essayer de répondre (même partiellement) à ces défis. J'en cite quelques unes (de façon non exhaustive) ci-dessous :

1.3.1. Elévation des niveaux d'abstraction

Depuis les dessins manuels de masques sur papiers, pour la conception des systèmes, il y a quelques années, les niveaux d'abstraction en entrée d'outils de conception ne cessent d'être élevés. Il s'agit d'une façon permettant de réduire (parfois de façon très significative) la complexité des spécifications manipulées par les concepteurs. Cette élévation des niveaux d'abstraction est toujours accompagnée de langages de haut niveau (souvent visuels) tels qu'UML permettant de modéliser des systèmes d'une grande complexité, de façon graphique.

1.3.2. Réutilisation de composants

Les SoCs modernes sont conçus à plus de 80% par réutilisation d'IPs. Qu'ils soient soft, hard, firmes ou logiciels, ces boites noires permettent aujourd'hui d'implémenter une grande partie des fonctionnalités logicielles et matérielles. Cette conception basée sur la réutilisation de composants est sans doute l'une des approches qui permettent aujourd'hui de réaliser des gains considérables en termes d'effort et de temps de conception. En effet, plus l'on utilise des boites noires dans un SoC, moins il reste de composants à concevoir à partir de zéro.

1.3.3. Utilisation des FPGAs

Un rapport de la société américaine Gartner Inc. indique que, d'ici à 2010, plus de 40 % des FPGA intégreront un microprocesseur embarqué [VIN09]. Ceux utilisés aujourd'hui permettent des vitesses de traitement de plus de 200 MHz pour les implémentations sur

processeur softcore et plus de 400 MHz pour les implémentations sur processeur hardcore. Avec de telles fréquences plus 80 % des applications embarquées nécessitant 32 bits peuvent bénéficier d'une solution FPGA. En plus de ces critères de performances les FPGAs modernes présentent de nombreux avantages tels que sa souplesse, la possibilité de les reconfigurer dynamiquement et leur faible coût d'intégration. Ces caractéristiques font des FPGAs la solution, par excellence, pour le prototypage.

1.3.4. Automatisation de la conception

L'ingénierie dirigée par les modèles (IDM) est considérée aujourd'hui comme l'une des approches les plus prometteuses dans le développement des systèmes. Cette technique offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène. Son objectif est de favoriser l'étude séparée des différents aspects du système. Cette séparation permet d'augmenter la réutilisation et aide les membres de l'équipe de conception à partager et s'échanger leurs travaux indépendamment de leur domaine d'expertise. L'IDM présente une forme d'ingénierie générative permettant d'aboutir au code du système décrit au haut niveau. Les outils de conception basés sur cette technologie me semblent donc aller dans le bon sens pour une automatisation maximale et sûre des plusieurs étapes de la production d'un SoC.

Mes recherches présentées dans ce mémoire se situent dans le contexte de la conception conjointe logicielle/matérielle de systèmes embarqués, plus particulièrement ceux implémentés sur FPGAs dynamiquement reconfigurables. Ainsi mes travaux de recherche, visent à apporter des éléments de solutions aux défis scientifiques de la conception de SoCs, en adoptant les solutions présentées précédemment, à travers un certain nombre de contributions. Une grande partie de mes travaux cible les systèmes implémentés sur FPGAs dynamiquement reconfigurables.

1.3.Contributions

Toutes mes donc concernent les méthodes et outils de conception conjointe logicielle/matérielle de systèmes embarqués, dédiés au traitement de signal intensif. Ainsi, mon objectif est de proposer des outils permettant de : modéliser, simuler et implémenter de tels systèmes de façon efficace et systématique.

Les flots de conception des systèmes embarqués commencent généralement par des descriptions de haut niveau (UML) pour aboutir à une implémentation physique sur ASIC ou FPGAs, en passant par l'inévitable étape de simulation.

Mes contributions couvrent ces trois principales phases de tout outil de conception. En effet, je me suis intéressé depuis 2002 (de haut en bas dans les niveaux d'abstraction, et non pas dans l'ordre chronologique) aux problèmes suivants :

- **Modélisation UML d'architectures dynamiquement reconfigurables :** il s'agit de modéliser tous les aspects liés à la reconfiguration dynamique à un très haut niveau d'abstraction (UML). Mes contributions sur cet aspect consistent en la proposition de méta modèles et profils UML, permettant de modéliser le contrôle dans les architectures reconfigurables et le modèle d'exécution au niveau RTL.
- **Co-simulation hétérogène, multi-niveaux, distribuée et estimation des performances et de la consommation d'énergie:** mes contributions sur ce problème consistent en un environnement de simulation, basé sur SystemC, supportant l'interaction

avec d'autres simulateurs tels que SpecC. Cet environnement supporte également les modèles de co-simulation distribués. Les communications et le transport des transactions sont assurés par des sockets ou le bus CORBA (ORB). Cette co-simulation prend en charge l'interaction entre composants décrits à des niveaux d'abstraction différents.

Le succès de tout système embarqué est étroitement lié à ses performances (en termes de temps d'exécution) et à sa consommation d'énergie. Les temps de mise sur le marché des nouveaux systèmes étant de plus en plus courts, il s'agit là de proposer des méthodes d'estimation non seulement précises mais aussi rapides. Deux différentes approches sont proposées pour répondre à ce problème. La première est manuelle, introduisant du code spécifique dans les IP de simulation. La seconde est non intrusive et est basée sur la notion d'espions de simulation.

- **Outillage et intégration dans un environnement de co-modélisation et de conception conjointe logicielle/matérielle : GASPARD :** Mes contributions en termes d'outillage, se situent à plusieurs niveaux dans la plateforme GASPAD. Ainsi au niveau simulation, Il s'agit de générer automatiquement le code de simulation SystemC à partir du modèle d'association de l'application et de l'architecture. Le code généré comprend l'interconnexion entre les composants de l'architecture, mais aussi la gestion de la synchronisation et du calcul d'adresses. Mes autres contributions consistent en la définition de la chaine de transformation modèle à modèle vers les descriptions VHDL RTL du système.

-

1.4. Organisation du manuscrit

Ce document est divisé en deux parties. La première est réservée à mes activités de recherche. Celle-ci est structurée en trois chapitres principaux. Chaque chapitre est terminé par une analyse et discussion.

Dans le premier, je présente mes travaux sur la modélisation au niveau UML des systèmes embarqués et en particulier ceux ciblant une implémentation sur FPGAs dynamiquement reconfigurables (contribution 1).

Dans le second chapitre, je détaille le travail que j'ai effectué sur la co-simulation hétérogène distribuée et sur l'estimation des performances et de la consommation de l'énergie par simulation. Dans ce chapitre, j'ai essayé d'abord de synthétiser les principales méthodes existantes avant de présenter mes contributions à la proposition de nouvelles méthodes (contribution 2).

Dans le troisième chapitre, je présente le projet GASPARD de l'EPI INRIA DaRT. Il s'agit d'un outil de co-modélisation et conception conjointe logicielle/matérielle de systèmes embarqués pour le traitement de signal intensif. Mes contributions en termes d'intégration et d'outillage sont présentées dans ce chapitre (contribution3).

La première partie de ce document est terminée par le chapitre 5 où un bilan général des différentes activités de recherche est présenté. Dans ce chapitre, mes projets de recherche pour les prochaines années (certains ayant démarrés en 2009 – 2010) sont aussi présentés. Il s'agit principalement de projets autour de la conception conjointe logicielle/ matérielle de systèmes à base de FPGAs partiellement et dynamiquement reconfigurables.

La seconde partie de ce mémoire présente mon curriculum vitae, incluant mes activités d'enseignement et responsabilités pédagogiques et administratives. Elle contient également quelques uns de mes articles scientifiques, détaillant certaines contributions présentées dans ce document.

Afin de produire un manuscrit cohérent, je me suis limité dans ce document à présenter uniquement les travaux de recherche entrant dans les trois thèmes : modélisation, simulation et outillage basé sur l'approche IDM pour la conception des SoC.

Volontairement donc, je n'ai pas repris certains travaux que j'ai réalisés notamment sur les réseaux d'interconnexion sur puce [AMA07a], [AMA07b], [MDS05], [QBM08] et sur l'optimisation du moteur de simulation SystemC [VMD04], [VPC05].

Chapitre 2: Modélisation

2.1. Introduction

Avec la croissante complexité des systèmes sur puce (en particulier les FPGAs), il est devenu impossible de les concevoir dans un bas niveau d'abstraction (comme le RTL) où il faut préciser chaque détail du comportement des composants. De ce fait, les ingénieurs se sont retrouvés face à un grand défi pour réussir à maîtriser cette complexité lors de la phase de conception de ces systèmes et d'arriver à une conception rapide sous de fortes contraintes de qualité et de temps de développement. Pour dépasser ce défi, ils ont fait appel à de nouvelles méthodes de conception basées sur des concepts d'abstraction de haut niveau ainsi que le raffinement, par transformation de modèles, jusqu'à atteindre le niveau bas.

Avec l'apparition récente des FPGAs supportant la reconfiguration partielle et dynamique, la modélisation «classique» de l'application et de l'architecture (comme dans tout flot basé sur un modèle en Y) ne semble plus être suffisante. En effet, ce type de matériel nécessite le pilotage de **la reconfiguration par le biais d'un contrôleur**, pouvant être implémenté en tant qu'accélérateur matériel ou comme fonction logicielle s'exécutant sur un processeur.

Nous ne disposons toujours pas, aujourd'hui, pour la synthèse du code d'implémentation, d'alternatives autres que les outils propriétaires. Cependant, l'entrée de ces outils (spécifications RTL), doit être produite par l'effort du concepteur. Ainsi, mettre à disposition, des concepteurs, **des méta-modèles permettant de générer de façon systématique et automatique de spécification RTL**, peut s'avérer d'une grande utilité. En effet, cela permettrait de réduire considérablement les risques d'erreurs dans la tâche fastidieuse de l'écriture manuelle de code RTL, ainsi que les temps de mise sur le marché des systèmes.

Dans mes travaux présentés dans ce chapitre, j'ai également essayé de dégager les concepts architecturaux dans les FPGAs, dynamiquement reconfigurables, existants dans les bas niveaux d'abstraction (RTL) et les faire remonter à un niveau d'abstraction plus haut. C'est-à-dire le niveau méta-modèle d'architecture. L'objectif étant de réaliser **une modélisation de FPGA qui se rapproche le plus possible de la réalité**, tout en se conformant (autant que possible) au standard MARTE. La modélisation proposée prend également en considération la problématique de **la consommation d'énergie des différents composants d'un FPGA**. En effet, la consommation doit être traitée très tôt dans le flot de conception afin d'estimer la performance du système et déterminer efficacement l'architecture la moins consommatrice d'énergie

2.2. Le contrôle dans les systèmes dynamiques

Le contrôle est un moyen permettant d'introduire la dynamicité dans un système embarqué. Cependant, les mécanismes de contrôle peuvent être intégrés de manières diverses et à différents niveaux dans le processus de conception conjointe logicielle/matérielle d'un SoC.

Dans les travaux je me suis intéressé à un type de contrôle particulier, dédié à la gestion et la supervision de la reconfiguration dynamique.

2.2.1. Etat de l'art

Concernant le contrôle, les approches actuelles de la gestion des reconfigurations impliquent:

- Des mécanismes d'observation des caractéristiques du système reconfigurable, comme par exemple, sa consommation d'énergie, ses qualités du service ou la charge de calcul,

- Des mécanismes de reconfiguration avec, par exemple, les liaisons/déconnexion des communications, changements d'IPs matériels et migration des tâches de calcul,

- Et également un contrôleur de reconfiguration qui ferme la boucle.

La conception d'un tel contrôleur s'appuie sur la programmation manuelle, et n'est toujours pas prise en charge par des outils et méthodologies de conception.

Les techniques de la théorie de contrôle peuvent être appliquées afin d'avoir de bonnes propriétés de la boucle de contrôle. En particulier, concernant le contrôle des modes de calcul, les systèmes à événements discrets et leur supervision [RaWo87]. Certaines méthodes liées à l'approche synchrone [HeMa00] ont été mises en œuvre dans des outils automatiques. Elles permettent de générer, au même niveau d'abstraction que la vérification par model checking, des contrôleurs « corrects ». Ces méthodes ont été intégrées dans une méthode de conception de systèmes logiciels [KA03], mais pas encore à ma connaissance, à la conception de circuits reconfigurables.

2.2.2. La contrôle dans un flot de conception en Y

Dans un flot de conception basé sur un modèle en Y, le contrôle peut être (traditionnellement) intégré aux niveaux :

- Application,

- architecture,

- allocation.

L'introduction du contrôle à l'un ou l'autre dans ces niveaux présente un certain nombre d'avantages et inconvénients. Ainsi, le principal défit auquel on se trouve confronté est le choix du niveau le plus approprié.

L'intégration du contrôle à chacun de ces niveaux, ainsi qu'une comparaison des différentes approches sont présentées dans la suite de cette section.

2.2.2.1. Le contrôle au niveau application

L'intégration du modèle de contrôle et de la construction d'automates de mode au niveau de l'application est très similaire aux automates de mode génériques de Gaspard décrits dans [Yu08a]. La figure 2.1 représentent un automate de mode au niveau application en illustrant un module de traitement d'effets de couleurs (ColorEffectTask) utilisé dans les téléphones

intelligents (smart phones). Ce module est utilisé pour gérer les effets de couleur d'un clip vidéo et propose deux options possibles (modes): couleur ou monochrome (noir et blanc), qui sont mises en œuvre respectivement par ColorFilter et MonochromeFilter. Ces deux filtres sont des tâches élémentaires au niveau de la modélisation de l'application, qui devraient être déployées sur des IPs. Le passage d'un filtre à l'autre est réalisé par ColorEffectSwitch lors de la réception des valeurs via son port de mode ColorMode. Les valeurs de modes sont déterminées par EffectController, dont le comportement est donné par son graphe d'état associé.

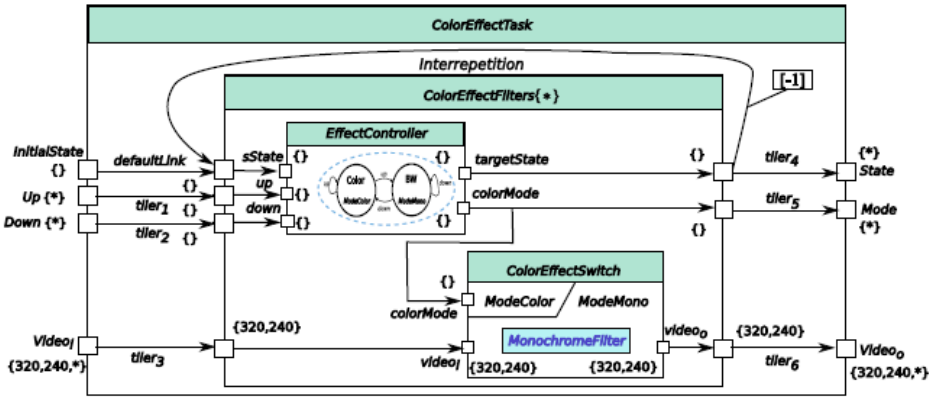


Figure 2.1: Un filtre de style de couleurs in modélisé avec un automate de modes Gaspard2

Le modèle de contrôle permet la spécification de l'adaptabilité d'un système au niveau l'application [QYJ10]. Chaque mode dans le switch peut avoir des effets différents en ce qui concerne les exigences de l'environnement du système ou de la plateforme même. Chaque mode peut aussi avoir des besoins différents en termes de mémoire, charge CPU, etc.

Les besoins et modifications de l'environnement/ plate-forme sont reçus comme des événements, et considérés comme des entrées du contrôle.

2.2.2.2. Le contrôle au niveau architecture

La sémantique de contrôle peut également être appliquée sur le niveau architecture, en utilisant une approche similaire à celle décrite précédemment pour le niveau application. Par rapport à l'intégration du contrôle dans d'autres niveaux de la modélisation (tels que l'application et l'allocation), le contrôle dans l'architecture est plus flexible, et peut être mis en œuvre sous plusieurs formes.

En effet, un contrôleur peut modifier la structure de l'architecture en question, comme la modification du réseau d'interconnexions par exemple. Cette structure peut être modifiée globalement ou partiellement. Dans le cas d'une modification globale, la reconfiguration est considérée comme statique et le contrôleur doit donc être implémenté à l'extérieur de l'architecture cible.

Si le contrôleur est présent à l'intérieur de l'architecture, la reconfiguration est partielle et pourrait donc être une reconfiguration dynamique partielle. Toutefois, le contrôleur peut être lié à la fois aux aspects structurels et comportementaux de l'architecture. Ceci est le cas lorsque qu'il s'agit par exemple d'une unité de commande intégrée dans une unité de traitement, comme dans [YBM02] et [IKH01]. Ces techniques permettent la réduction de la consommation d'énergie en baissant la fréquence ou la tension d'un processeur en cours d'exécution.

2.2.2.3. Le contrôle au niveau allocation

Le contrôle au niveau de l'allocation peut être utilisé pour diminuer le nombre d'unités de calcul actives, afin de réduire les niveaux de consommation d'énergie globale. Les tâches d'une application, exécutées en parallèle sur des unités de traitement peuvent réaliser les calculs désirés à une vitesse intéressante, mais parfois, au prix d'un niveau de consommation d'énergie élevé.

En effet, la modification de l'allocation peut produire un grand nombre de combinaisons et des résultats finaux bien différents (en termes de performances). Ainsi, une tâche peut être affectée à une autre unité de traitement qui consomme moins d'énergie, de même, toutes les tâches peuvent être associées à une seule unité. Cette dernière solution permet de réduire les niveaux de consommation d'énergie, mais avec diminution de la vitesse de traitement.

Le niveau allocation permet ainsi l'intégration de l'exploration d'architectures dans la conception. Cette exploration est réalisée par les concepteurs en fonction de leurs critères de qualité de service (QoS) retenus.

2.2.3. Comparaison de l'intégration du contrôle aux trois niveaux

L'intégration du contrôle dans chaque modèle du système (application, architecture et allocation) possède ses avantages et ses inconvénients, comme indiqué brièvement dans le tableau 2.1.

Dans l'intégration du contrôle, nous nous sommes intéressés à plusieurs aspects tels que l'étendue des effets sur les autres niveaux de la conception du SoC. Ainsi, nous définissons cette étendue comme étant soit locale ou globale. Dans le premier cas, l'impact ne concerne que le niveau de la modélisation en cause, alors que dans le second les conséquences concernent également d'autres niveaux de la modélisation.

Ces conséquences peuvent varier et provoquer des changements dans les deux aspects fonctionnels ou non fonctionnels du système. Ainsi, l'intégration du contrôle au niveau de l'application a un impact local (indépendant de l'architecture et de l'allocation.

L'intégration du contrôle dans l'architecture peut avoir plusieurs possibilités. En effet, le contrôle peut être principalement concerné par la modification des paramètres du matériel, comme la tension et la fréquence, pour des changements dans les niveaux de consommation. Ce type de contrôle est local et utilisé principalement pour la qualité de service. Un second type de contrôle peut être utilisé pour modifier la structure du système de façon globale ou partielle. Ce qui peut influer sur les autres niveaux de modélisation tels que l'allocation. En effet, l'allocation doit être modifiée chaque fois qu'il y a une modification dans la structure de la plateforme d'exécution.

Le contrôle au niveau allocation est local uniquement lorsque l'application et les modèles d'architecture ont été prédéfinis pour être de nature statique, or, ceci est rarement le cas dans les systèmes réels. Si l'application ou l'architecture est modifiée, l'allocation doit être adaptée en conséquence.

Il est également possible de former une sorte de contrôle fusionné, en combinant les modèles de contrôle sur les différents modèles du système, pour former une approche de contrôle multi-niveaux. Cependant dans ce cas, une analyse détaillée est nécessaire pour veiller à ce que toute combinaison de niveaux de contrôle ne cause pas de conséquences non désirées. Ceci est une tâche fastidieuse, car lors de l'analyse, plusieurs aspects doivent être surveillés, tels que le fait qu'aucun conflit ne surgisse en raison d'une approche multi-modèles. De même, la redondance doit être évitée: si un contrôle de l'application et celui de l'architecture produisent les mêmes résultats séparément; la suppression du contrôle d'un de ces niveaux serait justifiée. Toutefois, cela peut aussi conduire à une instabilité du système.

Il est également possible de créer un contrôleur global qui serait responsable de la synchronisation de divers mécanismes de contrôle locaux. Cependant, la sémantique doit être clairement définie pour la composition du contrôleur global, qui pourrait conduire à une augmentation de la complexité du système.

Le tableau 2.1 montre une comparaison globale d'intégration de contrôle aux trois niveaux.

	Impact sur les autres modèles	**Conditions**	**Conséquences**
Application	Local	_	Changement dans l'application (QoS éventuellement)
Architecture	Local	Variation des QoS	Variations dans les performances observées
	Global	Modification de la structure	Le modèle d'allocation doit être modifié
Allocation	Local	Les autres modèles sont fixes	Variations dans les performances observées
	Global	Les autres modèles sont modifiables	Changement dans la fonctionnalité (QoS éventuellement changées)

Tableau 2.1.: Intégration du contrôle dans les trois premiers niveaux de la conception de SoCs.

2.2.4. Le choix du niveau déploiement

L'impact global de tout modèle de contrôle n'est pas souhaitable dès lors que la modélisation des systèmes devient de plus en plus complexe et nécessite la gestion de plusieurs niveaux d'abstraction. Ainsi, une approche locale est plus souhaitable, car elle n'a aucune incidence sur les autres niveaux de modélisation. Cependant, dans chacun des modèles de contrôle mentionnés précédemment, des conditions strictes doivent être respectées pour leur intégration. Le respect de ces conditions peut s'avérer être une tâche très fastidieuse dépendamment de l'environnement de conception.

En effet, un modèle de contrôle idéal serait celui qui a une étendue d'impact uniquement locale sans conditions et contraintes strictes d'intégration.

Ainsi, un modèle de contrôle au niveau de déploiement semble être la solution idéale, car il aurait un impact local et complètement indépendant des trois niveaux présentés précédemment.

2.2.5. Le contrôle au niveau déploiement

Dans cette section, je présente l'intégration du contrôle à un autre niveau d'abstraction dans le flot de conception. Ce niveau permet de lier les tâches de l'application modélisés et les composants de l'architecture à leurs implémentations respectives.

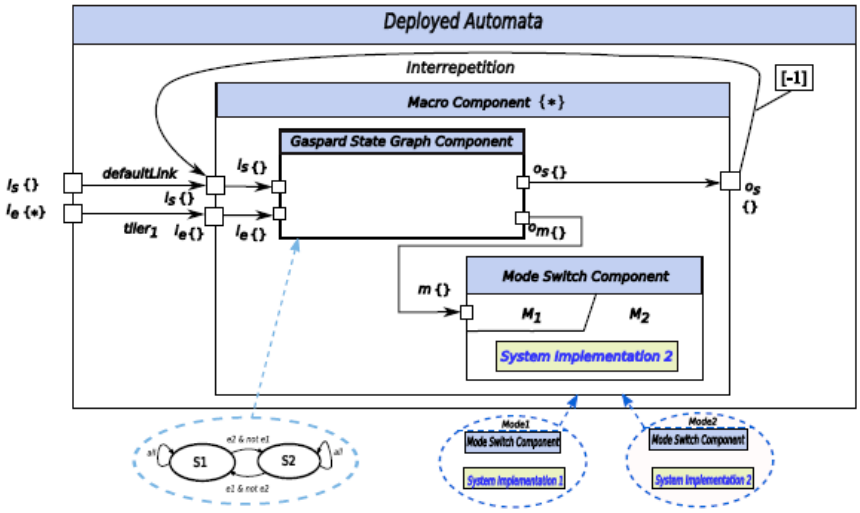


Figure 2.2: Intégration du contrôle au niveau déploiement

La Figure 2.2 montre l'intégration du contrôle au niveau déploiement dans un flot de conception conjointe logicielle/matérielle de SoC. Comparé à des modèles de contrôle à d'autres niveaux, qui ne tiennent compte que des aspects structurels de la conception, ce modèle de contrôle porte sur les aspects comportementaux.

Les automates au niveau déploiement, appelés Deployed Mode Automata traitent les composants élémentaires (atomiques) et leurs implémentations (présentes au plus bas niveau hiérarchique dans la modélisation).

Par rapport à d'autres modèles de contrôle, un mode dans un composant *mode switch* représente un système global qui est une collection de différentes implémentations locales, liées à leurs composants élémentaires.

Ainsi le flux de données associé aux automates de mode génériques de Gaspard n'est pas explicitement exprimé et les ports d'entrée/sortie sont supprimés à tous les niveaux hiérarchiques dans ce modèle de contrôle. Le graphe l'état Gaspard associé à ce modèle de contrôle ne porte que sur les implémentations globales du système. C'est pour cette raison que nous nous intéressons seulement aux graphes d'états mis à plat. Nous avons également besoin de traiter les événements entrants dans les Deployed Mode Automata.

Dans un modèle de contrôle au niveau application ou architecture, les événements arrivent soit à partir de l'environnement extérieur (par exemple un stimulus utilisateur) ou ils se produisent au hasard dans l'application ou l'architecture (en raison de l'action de certains composants élémentaires).

Toutefois, au niveau déploiement, les événements entrants ne sont pas liés à la modélisation de haut niveau, mais sont essentiellement utilisés pour représenter les entrées utilisateur de bas niveau, en fonction de la plate-forme d'exécution choisie. Par exemple au niveau RTL, ces événements utilisateur peuvent arriver sous la forme d'entrées de l'utilisateur ou l'environnement à partir d'une caméra attachée à un FPGA, ou d'entrées reçues via une UART.

Un concepteur modélisant le système à un haut niveau d'abstraction ne devrait pas se soucier de ces détails d'implémentation. Toutefois, afin de rendre ce modèle de contrôle aussi souple que possible, et de respecter la sémantique du modèle de contrôle abstrait, des ports d'événement ont été ajoutés à cette proposition. Ainsi, au cours de la transformation de

modèles et la génération de code, ces ports sont remplacés par des valeurs d'événements réels, qui sont utilisés pendant la phase d'implémentation sur FPGAs.

De même, pour les automates de mode au niveau application (ou architecture), son état initial est donné par un composant de l'application (ou architecture) possédant des ports événements d'entrée et un port de l'état de sortie.

Au début, certains événements sont générés comme entrées par ce composant afin de produire l'état initial. Après cela, le composant reste inactif en raison de l'absence d'événements arrivant sur ses ports d'entrée. Cet état initial est ensuite envoyé à l'automate de mode et servira à déterminer l'état initial du graphe d'états Gaspard. Cependant, pour un deployed mode automata, les aspects structurels sont absents et seulement les informations relatives aux composants élémentaires sont présentes. Ainsi, l'état initial lié au graphe d'état de déploiement Gaspard ne peut être déterminé de manière explicite.

Cette limitation a été supprimée par l'introduction de nouveaux concepts au niveau de déploiement, qui aident à déterminer l'état initial du deployed mode automata. Toutefois, la proposition maintient l'utilisation du port de l'état initial et du concept defaultLink, car ils aident à se conformer au modèle de contrôle abstrait, et sont utilisés dans les transformations de modèles ultérieures (pour une éventuelle génération de code).

Enfin, le contrôle proposé au niveau déploiement permet seulement la création d'une machine d'état pour un contrôleur de reconfiguration. Cependant, en cas de FPGA contenant plusieurs processeurs (logiciels ou matériels) embarqués, il est possible d'en choisir un seul pour faire office contrôleur. Mais, cela nécessite une modélisation de haut niveau appropriée, qui n'a pas encore été traitée dans mes travaux de recherche.

Actuellement, le code généré à partir de notre flot de conception est explicitement lié à un contrôleur générique, et il appartient à l'utilisateur de déterminer la nature et la position du contrôleur.

2.2.6. Avantages du contrôle au niveau déploiement

Le premier avantage d'utiliser le contrôle au niveau déploiement est que le niveau de l'impact demeure local et il n'y a pas d'influence sur les autres niveaux de modélisation. Un autre avantage est que l'application, l'architecture et les modèles d'allocation peuvent toujours être réutilisés, seules les IP nécessaires sont modifiés.

Bien que de nombreuses approches existent aujourd'hui pour exprimer la sémantique de contrôle, les automates de mode ont été choisis car ils permettent la séparation des flux de données et de contrôle. Ils sont également adaptés pour une approche basée sur les états, ce qui facilite une intégration dans notre environnement. Enfin ce type d'automate peut être exprimé au niveau de spécification MARTE.

En ce qui concerne la reconfiguration dynamique partielle, des différentes implémentations d'une région reconfigurable doivent avoir la même interface externe pour l'intégration avec la région statique, au moment de l'exécution.

La sémantique des automates mode de contrôle peuvent exprimer les différentes implémentations (modes/configurations) collectivement par le biais du concept de mode switch, qui peut être exprimé graphiquement à des hauts niveaux d'abstraction. De même, un graphe d'état spécifie le contrôleur responsable du passage entre les différentes configurations.

2.2.7. Extension du profil et méta-modèle MARTE

Le profil MARTE permet de modéliser des sémantiques comportementales UML. Cependant, en l'absence d'un environnement sous-jacent permettant d'interpréter ces modèles de haut niveau, les diagrammes de modèles ne peuvent être utilisés qu'à des fins de spécification. En outre, alors que le méta-modèle MARTE introduit la notion de comportement, les concepts associés qui permettent de concrétiser le comportement exact d'un système font défaut. Ainsi, en raison de ces manques dans le méta-modèle, il est extrêmement difficile d'interpréter les modèles de haut niveau, par les outils de transformation modèle à modèle, en vue d'une génération de code.

Il appartient donc à l'environnement de modélisation sous-jacent d'étendre le méta-modèle MARTE, pour permettre aux concepteurs de modéliser correctement un comportement spécifique. De même, les transformations de modèles doivent être élaborées en prenant en compte les nouveaux concepts, pour une génération de code éventuelle.

En outre, un certain nombre de problèmes se posent pour la modélisation des concepts comportementaux basés sur les états dans MARTE. Ainsi, nous nous sommes tournés vers les concepts UML de base, tels que les machines d'état et les collaborations [OMG03] pour les intégrer dans le méta-modèle existant, avec quelques modifications, afin de respecter la sémantique du méta-modèle MARTE et les transformations du modèle.

Dans la version étendue du méta-modèle MARTE pour l'intégration de notre modèle de contrôle, les machines d'états ont été choisies pour illustrer le comportement basé sur l'état d'un composant individuel, qui agit comme un élément de contrôle dans le système. Les machines d'états ont été adoptées parce qu'elles sont compatibles avec la modélisation basée sur les états. En outre, elles sont considérées comme le noyau essentiel pour un contrôleur de reconfiguration.

Comme dans Gaspard2, le contrôle et les calculs sont séparés, la modélisation de la structure du calcul contrôlée n'est pas suffisante. Ainsi, les collaborations sont utilisées pour illustrer le comportement des composants contrôlés dans le système.

Enfin, alors que le profil MARTE et son méta-modèle permettent la spécification de l'application, l'architecture et l'allocation d'un SoC, ils ne contiennent pas les concepts permettant de relier les composants élémentaires à leurs implémentations. Ainsi, les concepts additionnels de déploiement dans Gaspard2 doivent être intégrés dans le méta-modèle MARTE et son profil.

La suite de cette section présente les concepts ajoutés dans la version actuelle du méta-modèle MARTE; leur intégration conduisant à une version étendue de MARTE au moyen d'un mécanisme de fusion.

2.2.7.1. Limitations du niveau déploiement actuel.

Le concept de déploiement permet, jusqu'à présent, d'aider à créer une implémentation statique d'un système reconfigurable. Cela est dû au fait que, si l'on dispose de concepts permettant d'associer plusieurs implémentations à un composant élémentaire donné, les ambiguïtés surgissent quand le concepteur doit déterminer (et par la suite implémenter) une configuration du système. Cette configuration est à son tour en cause avec les composants élémentaires et leurs implémentations associées.

Par exemple, dans une application, un composant élémentaire P a trois implémentations P1, P2 et P3. Le concepteur peut sélectionner toutes ces implémentations et les associer à P, pour cibler un SoC reconfigurable. Il peut ainsi implémenter trois configurations du système ConfigurationQ, ConfigurationR et ConfigurationS, associées respectivement à P1, P2 et P3.

Toutefois, pendant la phase d'implémentation, il ne serait pas possible de déterminer quelle réalisation de P appartient à quelle configuration avec la sémantique du modèle de déploiement.

Comme le nombre de composants élémentaires et de leurs implémentations disponibles augmentent, la complexité augmente aussi de façon exponentielle. Ainsi, un mécanisme doit être mis en place pour résoudre cette ambiguïté de conception, présente dans la modélisation au niveau déploiement.

2.2.8.2. Notion de configurations.

Afin de répondre au problème mentionné ci-dessus, nous avons introduit la notion de configuration dans le méta-modèle de déploiement. Une configuration peut être spécifiée pour l'application, l'architecture ou peut être considéré comme une composition collective (mapping des deux aspects pour former un système global).

Dans l'état d'avancement actuel de mes travaux, seul le modèle d'application est pris comme entrée pour aboutir à l'implémentation finale sur FPGAs. Ainsi, nous ne considérons actuellement que les implémentations logicielles d'une configuration. Un IP logiciel peut donc faire partie d'une ou plusieurs configurations.

Une configuration possède les attributs suivants. L'attribut name permet de préciser le nom de la configuration (donné par le concepteur). L'attribut ConfigurationID permet d'attribuer une valeur unique pour chaque configuration, il est utilisé par les aspects de contrôle présentés au début de ce chapitre. Ces valeurs sont utilisées par un graphe d'états Gaspard pour produire les valeurs de modes associées à des composants Gaspard. Ces valeurs de modes sont ensuite envoyées à un composant de type mode switch, qui fait la correspondance entre les valeurs et les noms de ses collaborations. Si les valeurs correspondent, mode switch passe à la configuration requise. L'attribut InitialConfiguration définit une valeur booléenne pour une configuration afin d'indiquer s'il s'agit de la configuration initiale à charger sur le FPGA cible. Cet attribut permet également de déterminer l'état initial du graphe d'états Gaspard.

Ainsi, en combinaison avec les concepts de contrôle, le niveau de déploiement crée plusieurs configurations pour l'exécution finale (s) dans un FPGA. Chaque configuration est considérée comme un ensemble de différents IPs. Chacun de ces derniers est associé à son composant élémentaire respectif. Les transformations de modèles pour la chaîne RTL a ainsi été modifiée pour produire différentes implémentations d'un accélérateur matériel (chacune correspondant à une configuration spécifiée) dans un FPGA. Un composant élémentaire peut également être associé au même IP dans des configurations différentes.

Il est ainsi possible de relier plusieurs IPs avec le composant élémentaire correspondant, et chaque lien se rapporte à une configuration unique. Nous imposons la condition que pour tout nombre *n* de configurations, chacune ayant *m* composants élémentaires, tout composant élémentaire d'une configuration doit avoir au moins un IP associé. Cela permet la création efficace d'une configuration complète pour l'implémentation finale. Cette condition est déterminée par la référence de l'IP logiciel *softwareIP* à partir d'une configuration vers la classe du *SoftwareIP*.

La figure 6.3 représente une vue d'ensemble abstraite du mécanisme de configuration mis en place au niveau déploiement. Nous considérons une application Gaspard2 possedant trois composants élémentaires *EC X*, *EC Y* et *EC Z*, ayant respectivement comme implémentations IPX1, IPX2; IPY1, IPY2 et IPZ1. S'agissant d'une figure abstraite, j'ai omis délibérément de représenter certains concepts tels que *VirtualIP* et `Implements`. Cependant, cette représentation est très proche de la modélisation UML. Un changement dans

l'implémentation associée à l'un de ces composants élémentaires peut produire un résultat final différent concernant la fonctionnalité globale et différents critères de qualité de service.

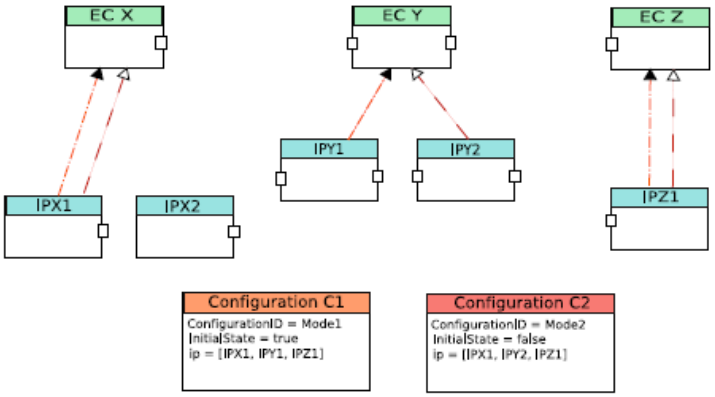


Figure 6.3: Vue abstraite de reconfigurations au niveau déploiement

Ici, deux configurations *Configuration C1* et *Configuration C2* sont illustrées sur la figure 6.3. *Configuration C1* est choisie comme la configuration initiale et a associé les IPs: IPX1, IPY1 et IPZ1. De même *configuration C2* possède aussi ses IPs associés. Cette figure illustre toutes les possibilités: un IP peut être globalement ou partiellement partagés entre les différentes configurations (comme IPX1), ou peut n'être inclut dans aucun cas (comme IPX2).

2.2.8. Thèses et masters

Les travaux autour du contrôle, dans l'équipe DaRT, ont été entamés avec les travaux de [LAB06] et [YU08a]. Cependant, les contributions présentées là proviennent essentiellement de la thèse [QUA10] que j'ai codirigé à 50%. Il s'agit d'un travail récent, et la thèse a été soutenue en avril 2010.

2.2.9. Publication représentative

Titre
High-level modeling of dynamically reconfigurable heterogeneous systems
Publication
Chapter 7 de: Heterogeneous Embedded Systems - Design Theory and Practice: Springer, 2010
Auteurs
Jean-Luc Dekeyser, Abdoulaye Gamatié, Samy Meftali et Imran Rafiq Quadri
Résumé
Ce chapitre présente une approche basée sur les composants de haut niveau, pour exprimer les SoCs ; basés sur la reconfiguration des systèmes dans un environnement de conception. Un modèle générique de contrôle réactif est définit pour l'outil Gaspard2. L'intégration du contrôle à différents niveaux du flot est explorée avec une comparaison de leurs avantages et inconvénients. Après l'intégration du contrôle, à un autre niveau d'abstraction élevé est étudiée, pour se révéler être plus avantageuse que les autres alternatives. Enfin, une étude de cas est présentée à des fins de validation. Les travaux présentés sont fondées sur l'approche Model-Driven Engineering (MDE) et le profil MARTE UML pour la modélisation et l'analyse des systèmes embarqués temps réel.

2.2.10. Analyse et discussion

Mes travaux sur la modélisation et l'intégration du contrôle dans la conception de SoCs, ont commencé par l'étude des aspects liés à la sémantique de contrôle. Ainsi, l'exploration des différents aspects de contrôle à différents niveaux dans la modélisation de SoCs (application, architecture et allocation), ont permis de choisir le niveau déploiement.

En effet, il ressort que le contrôle au niveau déploiement n'a qu'un effet local sur la modélisation, et n'affecte pas (ne remet pas en question) les modèles d'application et d'architecture. Il permet ainsi une grande réutilisation des modèles.

L'intégration de la sémantique de contrôle au niveau de déploiement dans la méta-modèle MARTE, permet l'expression des aspects liés à la reconfiguration dynamiques via les outils actuels de modélisation UML (supportant le profil MARTE). Le but de cette modélisation du contrôle étant de permettre des transformations automatiques (modèle à modèle) pour produire des spécifications de contrôleurs au niveau RTL, conformes au méta-modèle RTL, présenté dans la prochaine section.

Ces résultats obtenus aujourd'hui ne sont qu'un début de mes travaux de recherche liés au contrôle et à la dynamicité dans les SoCs modernes. La solution choisie permet effectivement d'aller vers des implémentations concrètes, cependant elle présente encore un certain nombre de limitations.

En effet, les contrôleurs que l'on génère actuellement se limitent à des reconfigurations mettant en œuvre les composants (matériels ou logiciels) du système et les IPs qui les implémentent. Cependant, le concept de la reconfiguration partielle et dynamique permet d'éventuels changements, dans le système, bien plus puissants que cela. Il est effectivement, tout à fait envisageable, dans les systèmes modernes d'avoir à faire à des reconfigurations faisant intervenir (en plus des aspects traités dans mes travaux) :

- Des modifications de l'architecture : avec éventuellement l'apparition (ou disparition) de certains composants de l'architecture matérielle, ou simplement des changements dans la topologie d'interconnexion ;
- Des changements non seulement dans le code d'implémentation des tâches mais aussi dans leur nombre et leur nature.
- Le passage d'un mode d'exécution à un autre (par exemple du mode pipeline au mode parallèle, ou parallèle vers séquentiel.. etc.)
- La migration de tâches logicielles en matériel et inversement (partitionnement dynamique.)

Modéliser et concevoir des contrôleurs prenant en charge tous ces aspects (de façon exhaustive) donneraient certainement lieu à des composants d'une grande complexité et très consommateurs d'énergie Ainsi, il me semble que cette problématique restera comme étant l'un des défis majeurs dans la conception de SoCs, à base de FPGAs dynamiquement reconfigurables, dans les années à venir. La principale difficulté sera certainement de trouver le bon compromis entre complexité, consommation et prise en charge des différents types de reconfiguration par le contrôleur.

2.3. Un méta-modèle ciblant le niveau RTL

Cette section présente le méta-modèle RTL de Gaspard2 (version 2 de l'environnement Gaspard). Il correspond à un niveau intermédiaire entre la modélisation d'une application de Gaspard2 (spécifié en utilisant le profil MARTE) ainsi que son déploiement, et la génération automatique de code, permettant l'implémentation finale sur un FPGA pour réaliser la reconfiguration dynamique partielle.

En raison de la présence de la sémantique de contrôle mis en place au niveau déploiement, la fonctionnalité du matériel a de multiples implémentations associées (équivalentes aux configurations modélisées au haut niveau), permettant la création d'un accélérateur matériel reconfigurable dynamiquement. En ce sens, le méta-modèle RTL prend en compte les détails du niveau de déploiement liés à cette transformation.

Le méta-modèle RTL peut être considéré comme un ensemble de concepts. Il contient quelques caractéristiques communes (méta-classes), utilisées pour la spécification des concepts liés à la création d'une fonctionnalité de matériel, tout en enrichissant de contrôle décrit dans la section précédente. En parallèle, il existe des méta-classes spécifiques et métarelations, chacune correspondant uniquement à l'un des aspects mentionnés ci-dessus.

Pour la description de la fonctionnalité du matériel, le méta-modèle doit être suffisamment détaillé pour que les transformations de modèles puissent générer du code synthétisable efficace pour une cible à base de FPGA. Une telle caractéristique rend également les transformations de code plus simples et faciles à coder.

2.3.1. Etat de l'art

Dans [WAH07], les auteurs présentent un modèle d'exécution de matériel basé sur Array-OL [BOU07], mais il est dépourvu de tous les aspects liés au parallélisme de tâches. Cependant, dans [LEB07b], le modèle d'exécution basé sur Array-OL permet l'expression des deux types de parallélisme tâches et données. Malheureusement, ces travaux ne prennent pas la reconfiguration dynamique en compte et l'accélérateur matériel ciblé est de nature statique.

Ainsi, le modèle d'exécution de matériel auquel je me suis intéressé dans mes travaux supporte le parallélisme potentiel des applications modélisées, tout en intégrant les fonctionnalités de reconfiguration dynamique. Ce modèle d'exécution ne concerne que l'application modélisée, et permet sa transformation en matériel ayant des caractéristiques dynamiques. En parallèle, les concepts de contrôle mentionnés au début de ce chapitre sont également intégrés dans le méta-modèle, pour décrire la sémantique et la syntaxe d'un contrôleur de reconfiguration. Ce contrôleur est finalement responsable de la gestion de la reconfiguration partielle du matériel produit. Ainsi, les concepts de contrôle RTL sont utilisés dans l'éventuelle génération de code pour le contrôleur de reconfiguration.

2.3.2 Le méta-modèle RTL

Pour les besoins de génération de code par transformation de modèles, une première version du méta-modèle RTL a été proposée dans [LEB07b]. Cependant, il n'intègre pas les aspects liés à la dynamicité. En fait, il était destiné à la création d'un accélérateur matériel statique unique pour l'implémentation final comme une boîte noire. La version actuelle du méta-modèle RTL développés au cours de mes travaux de recherche, permet l'intégration de fonctionnalités dynamiques, pour faciliter la création d'un accélérateur matériel reconfigurable dynamiquement.

Le méta-modèle prend en entrée, une collection de concepts liés au modèle d'exécution matérielle des applications Gaspard2, ainsi que les concepts de contrôle détaillés

précédemment. En bref, l'accélérateur matériel généré (et toutes ses implémentations disponibles) présente un certain nombre de caractéristiques comme la hiérarchie et le parallélisme de données et de tâches spécifiés dans le modèle d'application de haut niveau en UML. La partie du méta-modèle liée aux accélérateurs matériels est totalement indépendante de toute syntaxe spécifique d'un HDL. Cependant, son bas niveau d'abstraction permet une génération de code, assez systématique, pour un HDL cible. De même, les concepts de contrôle enrichi peuvent être interprétés pour la génération de code HDL dans le cas d'un contrôleur matériel, ou un code de plus haut niveau comme C/C++ pour l'exécution du contrôleur sur un microprocesseur.

Étant donné que mes travaux de recherche visent principalement l'auto reconfiguration interne partielle dynamique, la solution basée sur les microprocesseurs a été adaptée.

Dans cette section, j'explique brièvement les concepts du méta-modèle RTL, et ce que l'on peut modéliser en l'utilisant. Cependant, la phase de génération de code est abordée dans le chapitre 4.

2.3.3. Pourquoi développer un méta-modèle RTL ?

2.3.3.1. Nature indépendante des plateformes

Le méta-modèle RTL possède tous les avantages classiques de l'utilisation des méta-modèles en général dans une approche IDM. En effet, il est de nature générique, et permet la réutilisation de concepts clairement définis (méta-classes) et leurs relations (métarelations).

Toutefois, dans le cadre de la Gaspard2, le méta-modèle RTL est un niveau intermédiaire entre le modèle UML d'une application déployée intégrant du contrôle et la phase de génération de code.

Ainsi, le modèle RTL est indépendant du langage cible (en particulier C/C + + pour les aspects de contrôle et de langages HDL, tels que VHDL ou Verilog, pour les accélérateurs matériel/contrôle).

2.3.3.2. Influence du méta-modèle MARTE.

Le méta-modèle RTL s'inspire du méta-modèle MARTE. Ainsi, les concepts que l'on trouve dans le paquetage RSM, liés au caractère multidimensionnel, ont été intégrés dans ce méta-modèle. De même des concepts tels que les composants, les ports et connecteurs du paquetage MARTE GCM ont également été traduits en tant qu'éléments quasi équivalents. Toutefois, le méta-modèle MARTE (ou même notre méta-modèle étendu MARTE) ne fournit pas une sémantique détaillée pour la génération d'un circuit intégré au niveau RTL. En effet, les descriptions RTL nécessitent un certain nombre de détails liés aux plates-formes d'exécution qui ne sont pas (et ne devrait pas exister) dans les méta-modèles de haut niveau.

2.3.4. Un aperçu du méta-modèle RTL

Dans le méta-modèle RTL, j'ai choisi de ne pas présenter tous les moindres détails liés au méta-modèle, mais plutôt de me concentrer uniquement sur les points clés.

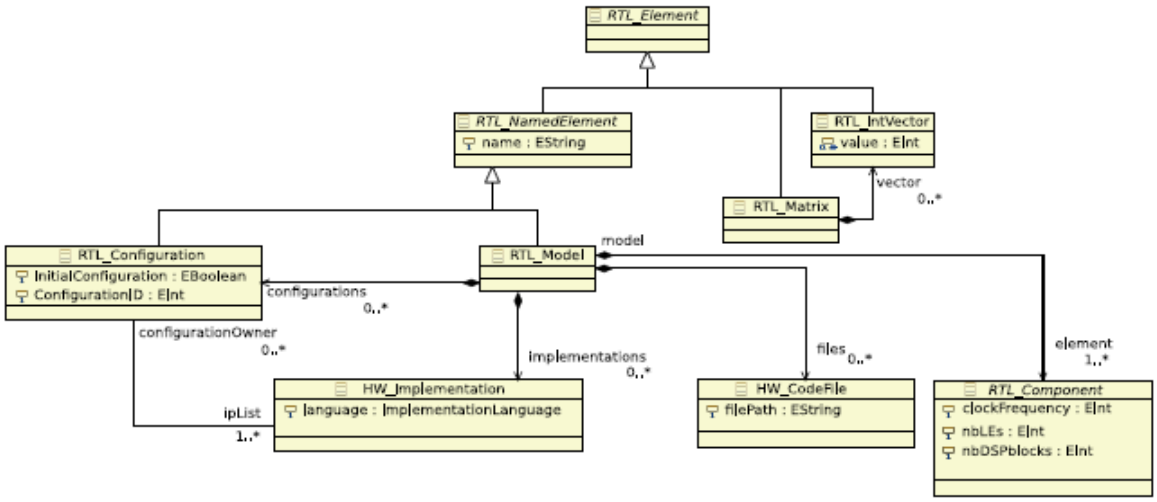


Figure 2.3: Noyau du méta-modèle RTL

Je présente tout d'abord les concepts de base du méta-modèle RTL. Le concept du plus haut niveau, présent dans le méta-modèle RTL, est le *RTL_Element* méta-classe qui représente une entité générique abstraite. La classe *RTL_NamedElement* est spécialisée à partir de ce concept et fournit un nom unique par le biais de son attribut associé, qui attribue un nom à chaque entité.

Le *RTL_Model* est l'un des modèles de sortie produits par la transformation *MARTE2RTL*. Cette méta-classe englobe tous les autres concepts présents dans le méta-modèle RTL, à l'exception des concepts liés aux types de ports, qui sont produits dans le modèle *RTL_PortType*. Un *RTL_Model* contient principalement *RTL_Component (s), HW_CodeFile (s), HW_Implementation (s), RTL_Configuration (s)* ainsi que des vecteurs et des matrices d'entiers: *RTL_IntVector* et *RTL_Matrix.*

Un RTL_Component est un composant générique abstrait dans le méta-modèle RTL: il peut être utilisé comme composant de base de l'accélérateur matériel ou un module de base pour le contrôle. Cette entité peut être hiérarchique ou élémentaire. Les concepts de *HW_CodeFile* et *HW_Implementation* ont des significations équivalentes à leurs appellations. De même, la notion de configuration dans le méta-modèle déploiement se traduit par une méta-classe *RTL_Configuration* équivalente. Un *RTL_Model* peut contenir des fichiers de code, implémentations et des configurations. Un *RTL_Matrix* représente une matrice spécifiée dans le modèle UML de haut niveau et se compose de *RTL_IntVector (s)* au moyen de la relation *vector*.

2.3.5. Analyse et discussion

J'ai présenté dans cette section le méta-modèle RTL développé et intégré dans l'environnement Gaspard2. Ce méta-modèle correspond à un niveau intermédiaire entre la modélisation d'une application Gaspard2 (spécifiée en utilisant le profil MARTE) ainsi que son déploiement et la génération automatique de code, permettant l'exécution sur un FPGA ciblé.

En raison de la présence de la sémantique de contrôle, mis en place au niveau du déploiement, la fonctionnalité du matériel transformé a de multiples implémentations associées (équivalente aux configurations modélisées au haut niveau), permettant la création d'un accélérateur matériel reconfigurable dynamiquement. Ainsi, le méta-modèle RTL prend en compte les détails du niveau déploiement liés à cette transformation, pour supporter la reconfiguration dynamique partielle.

Le méta-modèle RTL peut être considéré comme un ensemble de différents concepts. Il contient quelques unes des méta-classes utilisées pour la spécification des concepts liés à la création d'une fonctionnalité du matériel, tout en enrichissant le contrôle du niveau déploiement.

Il est, à mon sens, clair qu'un tel méta-modèle soit d'une importance capitale dans un environnement de conception, basé sur l'IDM et ciblant les FPGAs dynamiquement reconfigurables, tel que Gaspard2. En effet, c'est le seul moyen permettant de cibler ces architectures en générant, de façon systématique, du code RTL synthétisable pour l'entrée des outils commerciaux.

Il s'agit d'un méta-modèle qui augmente la flexibilité de Gaspard2, dans le sens où il reste valable pour cibler d'éventuels futurs langages de description au niveau RTL (moyennant une transformation modèle vers texte assez systématique). Il me semble donc que la poursuite de l'effort de recherche dans le même sens serait tout à fait pertinente.

Ainsi, la voie naturelle de mes travaux futurs sur cette problématique sera l'enrichissement de ce méta-modèle RTL pour qu'il puisse fournir, par transformations, des implémentations supportant les concepts de reconfiguration dynamique cités dans la section 2.2.9.

2.4. Modélisation de FPGAs et représentation de la consommation

Le standard MARTE (*Modeling and Analysis of Real-Time and Embedded systems*) dont la version préliminaire a été approuvée en juin 2007 a pour but d'apporter une unification «syntaxique» des différentes initiatives.

Le standard OMG MARTE est un profil UML qui définit les fondations de la modélisation des SETRs (Systèmes Embarqués Temps Réels) et est conçu pour succéder à son prédécesseur SPT [OMG05]. En effet, il présente de nombreux concepts permettant de modéliser et d'annoter un système logiciel/matériel en vue de l'analyser. L'objectif n'est pas de définir de nouvelles méthodologies de conception ou de nouvelles techniques d'analyse des SETRs, mais il s'organise en différents paquetages afin de les soutenir par une riche base concepts et d'annotations.

Malgré toute cette richesse, il semble que le profil MARTE (dans son état actuel) soit dépourvu de concepts suffisants pour modéliser les architectures basées sur les FPGAs dynamiquement reconfigurables. Ce manque s'étend également aux annotations permettant la modélisation de la consommation d'énergie dans les modèles de haut niveau.

Comme les modèles de Gaspard2 sont conformes à MARTE, il est d'une grande importance pour l'équipe DaRT de disposer dans ce profil de tous les concepts nécessaires à la modélisation des architectures qu'elle cible et qui sont les FPGAs. Ainsi j'ai commencé récemment à m'intéresser à ce problème et les résultats que je présente dans cette section correspondent donc à des recherches en cours.

2.4.1. Représentation de FPGAs et expressions des placements

Dans la littérature, les FPGAs sont représentés de manière à exprimer des placements de tâches et à fournir des caractéristiques sur les implémentations de ces tâches.

Selon l'objectif de la représentation, la précision et le niveau de détails de la modélisation des FPGAs peuvent varier. Avec l'orientation des représentations vers l'hétérogénéité, L. Singhal et *al.* [SiBo07] se sont rapprochés des FPGAs de type Virtex [XIL07] de Xilinx et Stratix de Altera.

S. Chaudhuri et *al.* [CDG07] ont étendu le VPR afin qu'il soit exploité en tant que modeleur et qu'il permette la génération d'une description VHDL du FPGA. Ceci est possible vu que l'outil VPR fonctionne dans un niveau de description des FPGAs assez proche du niveau RTL. Lagadec *et.al* [LLF01] introduisent le concept d'architecture FPGA virtuelle, similaire au concept de machine virtuelle. Cette vue virtuelle correspond à la vision qu'ont les applications du FPGA lors de leur implémentation par un OS, similairement à une mémoire virtuelle. Ainsi, les ressources physiques du FPGA sont regroupées en ensembles de ressources. Cette virtualisation permet de considérer le FPGA plus simplement, avec un grain correspondant à celui des applications qu'il implémente. L'objectif de la vitalisation est la définition d'une structure stable et portable des FPGAs virtuels pour le placement et le routage.

Le standard MARTE [FEB06] permet la représentation, en partie, des FPGAs mais se contente d'une classification en fonction de l'agencement des cellules reconfigurables. Il est à noter que la description de FPGAs hétérogènes n'est pas possible et aucune information ne permet de définir les entrées sorties du FPGA.

Selon MARTE, les FPGAs sont classifiés en fonction de la technologie de configuration, du type d'organisation, etc. Cependant, la classification proposée est

intéressante si la modélisation se fait à un très haut niveau d'abstraction (où la description du FPGA ne nécessite pas beaucoup d'informations) mais n'est pas exploitable telle qu'elle pour modéliser les placements.

En effet, il existe certains détails dans l'architecture de FPGA tels que les informations sur les Blocs Logiques Configurables (*Configurable Logic Block « CLB »*), les blocs d'Entrée/Sortie (*Input Output Block « IOB »*) et les Blocs RAM (*Block Random Access Memory* « *BRAM* »), qui appartiennent au niveau RTL, et ne sont pas forcément essentiels dans l'une ou l'autre des étapes de la conception. En utilisant les concepts de MARTE, il est impossible de faire remonter ces détails à un niveau d'abstraction plus haut (niveau méta-modèle d'architecture) afin de les modéliser.

De manière générale, Une représentation plus abstraite (et donc plus simple) est plus difficilement exploitable et elle nécessite l'utilisation des fichiers de contraintes par exemple. Il s'avère donc utile de générer une représentation unifiée et générale des FPGAs en fonction de leurs objectifs et du niveau de détails souhaité.

2.4.2. Evaluation de la consommation des FPGAs dans MARTE

[BEN08] présente une approche et outils permettant une estimation de la consommation statique et dynamique de l'énergie. Ces outils sont destinés pour être utilisables essentiellement au niveau CABA. Une méthode pour la modélisation de la consommation a également été proposée dans [Ben08], cependant elle n'utilise pas le standard MARTE mais a fait appel au profil UML.

Notons que, dans notre cas, le modèle énergétique de MARTE, ne permet de modéliser que la consommation d'un composant selon seulement deux états: Actif (*operating*) ou en repos (*storage*). Avec ces concepts, nous ne pouvons pas connaître les détails des activités des composants (types et consommation). Cette contrainte ne permet pas de localiser l'unité et les activités qui consomment le plus.

2.4.3. Extension du méta-modèle et prise en compte de la consommation

Comme déjà mentionné dans les paragraphes précédents, il y a certains détails (BRAM, CLB, etc.) dans l'architecture de FPGA qui ne peuvent pas être modélisés avec MARTE. Pour remédier à ce problème, nous pouvons ajouter les informations concernant ces composants en tant qu'attributs dans la classe FPGA. Nous pouvons ainsi mentionner, par exemple, le nombre de blocs logiques (CLB) qui existent dans le FPGA ou la taille des Blocs RAM (BRAM), etc.

Un des objectifs de mes travaux est de proposer une extension au niveau méta-modèle d'architecture de MARTE qui permettrait de prendre en considération la problématique de consommation d'énergie des composants dans FPGA.

MARTE ne permet pas d'identifier les différentes activités qui peuvent être exécutées par un composant. Par conséquent, nous ne pouvons pas connaître les ressources qui consomment le plus et le concepteur ne pourra donc pas proposer des solutions architecturales efficaces afin de diminuer la consommation.

Il faut donc, d'une part, définir les différentes activités propres à chaque composant et d'autre part, connaître la consommation de ces activités. Ceci nécessite de trouver une relation entre le modèle physique (HRM-h*wComponent*) et le modèle d'énergie (HRM-*HwPower*).

La solution proposée dans ces travaux est d'étendre le méta-modèle d'architecture de MARTE en ajoutant, au paquetage « *HwComponent* » un attribut « *activConsumption* » (c'est-à-dire la consommation de l'activité) de type « *NFP_Power* ».

Après avoir appliqué le profil MARTE sur le composant (*hwProcessor* et *hwComponent* des sous-ensembles respectifs logique et physique du HRM), nous pouvons identifier les différents types d'activités exécutées par le processeur comme: les opérations arithmétiques, logiques, de comparaison, etc. Par la suite, grâce à la propriété non fonctionnelle « *NFP_Power* » nous pouvons préciser la valeur de la consommation pour chaque activité. La figure 2.4 illustre la modélisation de la consommation d'un PowerPC.

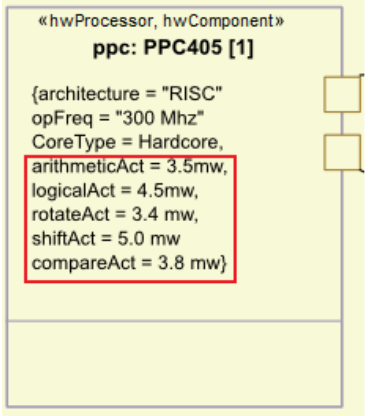


Figure 2.4: Exemple d'activités d'un composant avec leurs consommations

2.4.4. Modélisation des FPGAs

Dans le but de proposer un modèle général de FPGA, il a fallu étudier plusieurs exemples d'architectures afin d'essayer de déterminer les composants principaux qui constituent un FPGA (de façon générale).

Comme mentionné précédemment, les concepts de MARTE permettent de caractériser l'organisation de cellules de FPGA (lignes et colonnes) ainsi que le type d'architecture. Néanmoins ces concepts ne sont pas suffisants pour une description de FPGA à un très haut niveau d'abstraction car ils n'intègrent pas certains aspects tels que le type d'implémentation du processeur (logiciel ou matériel), type du bus, etc. Ces détails sont nécessaires pour modéliser une représentation complète d'un FPGA hétérogène.

De manière générale, nous distinguons quatre composants essentiels pour une représentation générale de FPGA qui sont: processeur, mémoire RAM ou cache, bus, arbitre, et éventuellement pont (bridge) pour l'utilisation de bus de différents types. Une modélisation a ainsi été proposée pour chacun de ces types de composants.

2.4.4.1. Processeur

Les concepts de MARTE liés à la représentation d'un processeur ne sont pas suffisants pour un SoC complexe, dans lequel un processeur peut être implémenté comme un «*softcore*» ou intégré comme un « *hardcore* ».

Pour résoudre ce problème, nous ajoutons l'attribut « CoreType » qui permet de spécifier le type d'implémentation du coeur. Il est soit un « *Hardcore* », soit un « *Softcore* », soit « *other* » s'il s'agit d'une autre technologie existante mais qui n'est pas encore spécifiée pour le moment, ou bien « *Undefined* » (indéfini) pour une éventuelle évolution du matériel et permettre une modification facile du modèle (figure 2.5)

Figure 2.5 : Le modèle du processeur proposé

Pour connaitre la consommation du processeur, nous considérons les activités arithmétiques (*arithmetic*) et logiques (*logical*) en tant qu'activités communes à tout type de processeur. Le processeur a une consommation même quand il est au repos (*idle*). Dans le cas où il est déconnecté du réseau, on dit que le processeur est « *inactif* ».

2.4.4.2. Mémoire

La mémoire dans les systèmes embarqués est responsable d'une grande partie de la consommation totale [MEF02]. Cette source de consommation s'amplifie avec les applications de traitement de données intensif dues à la grande quantité de données transitant entre les mémoires et les processeurs.

Ainsi, nous considérons que la mémoire consomme lorsqu'elle reçoit une requête de lecture (*read*), ou bien une requête d'écriture (*write*) ou même quand elle est au repos (*idle*).

2.4.4.3. Bus

Il est nécessaire de spécifier le type du bus dans une représentation de FPGA. Pour ceci, nous avons ajouté l'attribut « *BusType* ». Le bus peut être soit un bus *OPB* (*Onchip Peripheral Bus* pour la connexion entre les périphériques), soit *PLB* (*Processor Local Bus* pour la connexion entre le processeur et d'autres périphériques) (sachant que ces deux bus sont utilisés dans les FPGAs de Xilinx) soit *Avalon* (c'est un bus utilisé par Altera) soit «*Other* » et « *Undefined* » pour une extension et une flexibilité de choix.

Nous avons également besoin de connaitre le nombre de périphériques connectés au bus en mode maître (*nbMasters*) ou en mode esclave (*nbSlaves*).

Etant donné que le bus consomme seulement dans deux cas : quand il transmet des données c'est-à-dire en exécution (*operating*) ou bien quand il est au repos (*storage*), nous pouvons donc garder les concepts prédéfinis de MARTE.

2.4.4.4. Mémoire Cache

La mémoire cache est habituellement intégrée dans le composant processeur. Dans certains environnements, ce composant est représenté par un modèle séparé du processeur. Dans ce cas, l'architecture du cache utilisée (taille, associativité, stratégie d'écriture, etc.) influe sur les caractéristiques du système en terme de performances, surface et consommation d'énergie [NiMe04].

La séparation entre le modèle du processeur et celui de la mémoire cache permet d'obtenir une modularité de la conception. Ceci offre la possibilité d'explorer et d'évaluer différentes architectures. Ainsi, les activités principales identifiées pour un cache sont :

- *Read Hit* (succès de lecture) : le cache envoie une réponse valide pour la requête de lecture du processeur.

- *Read Miss* : il s'agit d'un défaut de cache dans la lecture de donnée (donnée n'existe pas).

- *Uncached read* : Le processeur peut demander la lecture d'une donnée qui ne peut pas être mise en cache. C'est le cas des données qui sont partagées par plusieurs processeurs et qui sont accédées en écriture.

- *Write Hit* (succès d'écriture) : le cache envoie une réponse au processeur pour confirmer l'écriture de la donnée.

- *Write Miss* : il s'agit d'un défaut de cache dans l'écriture de donnée.

- *Idle* : le cache est en *standby*.

2.4.4.5. Arbitre

Il s'agit du composant qui permet d'allouer le contrôle du bus aux périphériques qui le demandent, évitant tout conflit.

Il existe plusieurs stratégies d'arbitrage [Ben08], [SOC06] basées sur différents critères. Ainsi, la modélisation proposée permet de les spécifier, mais aussi de prendre en compte d'éventuelles stratégies futures.

2.4.4.6. Le modèle général proposé

La figure 2.6 présente le modèle général de FPGA proposé dans ces mes travaux. Tous les détails correspondants à la modélisation des différents composants, sont fournis dans [CHA09].

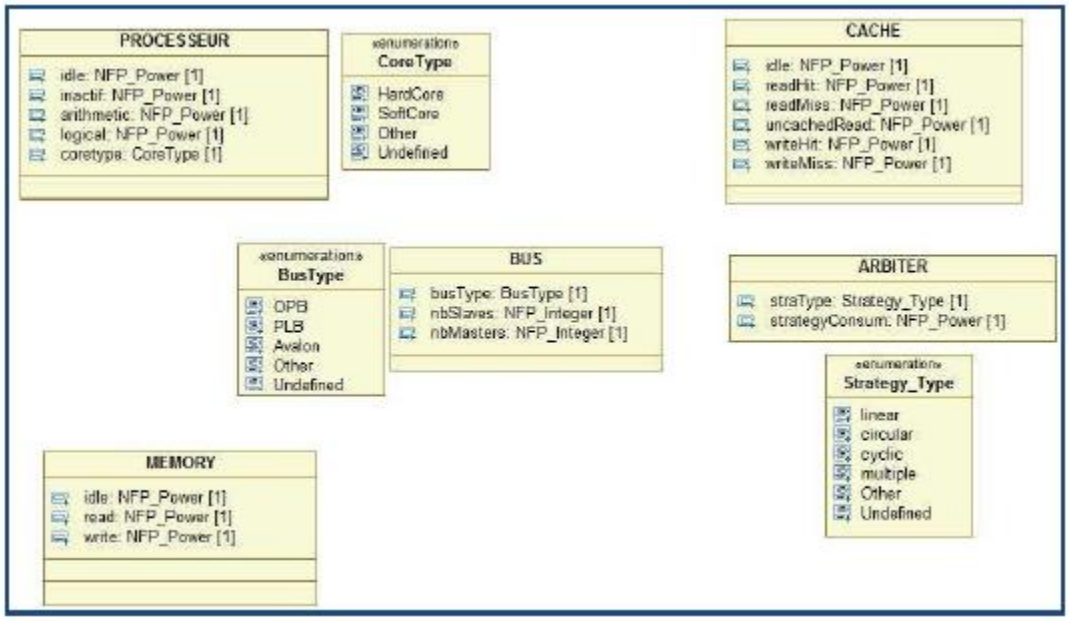


Figure 2.6. Modèle général proposé

2.4.4.7. Avantages du modèle proposé

Ce modèle permet aussi de dégager les différentes activités des composants dans une architecture FPGA et de spécifier la valeur de leur consommation. Cette modélisation permet d'intégrer certains détails au matériel reconfigurable, au niveau du modèle d'architecture. Il permet ainsi non seulement de prendre en compte le contrôle, mettant en œuvre des aspects matériels, dans les travaux futurs, mais aussi de prendre en compte la consommation d'énergie dés les premiers niveaux de modélisation.

Les valeurs correspondantes aux consommations des différentes activités de chaque composant ont été mesurées, sur plusieurs types de FPGAs en utilisant les outils Xilinx et ALtera. Les détails sur ces mesures sont donnés dans [BED09].

2.4.4. Thèses et masters

Ce travail sur la modélisation des FPGAs et leur consommation d'énergie, en utilisant MARTE est très récent. Il a démarré en 2009 avec les stages de master [CHE09] et [BED09] que j'ai encadré à 100%. Il s'agit d'efforts qui seront poursuivis dans le cadre de la thèse de Sana Cherif, démarrés en fin 2009, dans le cadre du projet ANR Famous et que je codirige à 50%.

2.4.5. Analyse et discussion

Dans sa version actuelle, le profil MARTE ne dispose malheureusement pas de concepts suffisants pour modéliser des architectures FPGAs dynamiquement reconfigurables. Ceci aurait surement été acceptable il y a cinq ans. Par contre, avec l'engouement que suscite de nos jours ce type d'architectures pour le prototypage notamment, un tel manque constitue un handicap considérable pour ce grand standard.

La consommation d'énergie est un critère important et constitue une contrainte forte dans tout système embarqué. Cette contrainte est encore plus accentuée, dés lors qu'il s'agisse d'un système à base de FPGAs dynamiquement reconfigurables. En effet, le concepteur de ce type de systèmes se doit d'être extrêmement vigilant quant à la consommation d'énergie due notamment au passage d'une configuration à une autre.

J'ai commencé très récemment à m'intéresser, dans mes travaux de recherche, à la modélisation des FPGAs dynamiquement reconfigurables, en prenant en compte la consommation énergétique. Ainsi, une approche basée sur MARTE (étendu) à été proposée pour modéliser les activités pertinentes de chaque composant d'un FPGAs (processeur, mémoire, bus, etc.) ainsi que leurs consommations d'énergie.

Il me semble cependant que beaucoup reste à faire dans ce domaine. En effet, on peut raisonnablement penser que les constructeurs de FPGAs, concurrents de Xilinx, comme Altera par exemple, se mettront dans l'avenir proche à la production de solutions FPGA reconfigurables. Ces nouveaux systèmes seront sans doute de plus en plus complexes et permettront l'implémentation de grosses applications gourmandes en puissance de calcul et en besoins de mémorisation.

La conception de ces systèmes ne pourra alors être envisagée qu'à partir de modèles de haut niveau, s'appuyant sur des standards tels que MARTE. L'effort d'enrichissement de ce standard par des concepts permettant la modélisation des architectures dynamiquement reconfigurables, me semble donc aller dans le bon sens.

Il faudra ainsi accorder une grande importance à la modélisation des FPGAs aux niveaux architecture et application, car sans elle il me parait très difficile d'imaginer des mécanismes indispensables dans l'exploration d'architectures efficaces, comme le partitionnement logiciel/matériel dynamique.

Chapitre 3: La simulation pour l'estimation des performances et de la consommation d'énergie.

3.1. Introduction

Les dernières années ont connu une grande évolution dans la technologie de fabrication des circuits intégrés. En effet, elles ont été marquées par l'apparition des SoC. Ces derniers sont de plus en plus complexes et intègrent des parties matérielles dédiées ou spécifiques, mais aussi des parties programmables de type processeur par exemple.

La plupart des outils de co-design commencent par deux spécifications séparées (de haut niveau d'abstraction) de l'application et de l'architecture. Ainsi, le flot de conception consiste à raffiner ces spécifications pour obtenir des descriptions, de plus en plus détaillées du système. Dans ces dernières, l'application et l'architecture peuvent rester séparées pendant certaines étapes, mais l'application est toujours déployée sur l'architecture plus tard dans le flot de conception.

Ainsi, avant d'obtenir un système monopuce sur du silicium, le concepteur doit le spécifier à plusieurs niveaux d'abstraction (fonctionnel, logique, transfert de registres,.. etc). Chacune de ces spécifications est appelée modèle. Tout flot de conception de systèmes consiste donc à raffiner plus ou moins automatiquement chaque modèle pour en obtenir un autre [JER02, JER07], et ce , en partant généralement du niveau fonctionnel jusqu'au niveau transfert de registres (RTL). Par conséquent, tout outil de conception doit comprendre impérativement un module de simulation permettant de valider les modèles avant de les raffiner. Pour cette raison, l'un des plus grands paris des concepteurs de nos jours est donc le développement d'un outil de simulation rapide et efficace permettant une vérification fiable des systèmes à chaque niveau d'abstraction.

Vu la complexité des systèmes sur puce modernes, ils sont de plus en plus conçus par assemblage de composants virtuels réutilisables (IP : Intellectual property). Ainsi, les concepteurs veulent souvent ajouter une certaine fonctionnalité à un système constitué d'un ensemble d'IPs préconçus.

Dans le cas, où le système est déjà disponible, par exemple, uniquement au niveau RTL, il peut s'avérer très bénéfique en terme de temps et d'effort de conception de concevoir seulement la partie que l'on veut lui rajouter, à partir du niveau fonctionnel, sans pour autant réécrire toutes la spécification de l'application à des hauts niveaux d'abstraction. Par conséquent, il devient nécessaire de disposer d'un outil de simulation permettant de valider le nouveau système (constitué de la partie décrite au niveau RTL et de celle décrite au niveau fonctionnel). Un tel outil doit donc permettre de simuler dans un même système des IPs décrits à différents niveaux d'abstraction (tous les niveaux utilisés dans le flot de conception). Ce type de simulation est dit **simulation multi-niveaux.**

L'hétérogénéité des SoCs actuels fait qu'ils soient le plus souvent conçus par plusieurs équipes d'ingénieurs, géographiquement éloignées, ayant chacune ses langages d'expertises (adaptés à des types de composants spécifiques) et ses propres outils de simulation. Ainsi, la validation du système entier, ne peut se faire qu'en utilisant un outil supportant la co-simulation hétérogène en terme de simulateurs, mais aussi l'exécution distribuée (sur des machines distantes) du modèle de simulation global. Ceci est appelé la **co-simulation hétérogène, géographiquement distribuée**.

La mise en œuvre d'une co-simulation, efficace, des SoC nécessite l'intégration d'outils **d'estimation de performance et de consommation d'énergie**. En effet, ces outils sont utiles pour une comparaison fiable et rapide des alternatives architecturales.

De telles estimations peuvent s'avérer particulièrement efficaces quand elles sont réalisées à des niveaux d'abstraction relativement élevés. En effet, la simulation à ces niveaux

est assez rapide pour permettre l'utilisation des résultats de performance dans des boucles d'exploration d'architecture rapide.

Ce chapitre présente l'état de l'art relatif aux trois problèmes identifiés, puis introduit mes contributions pour chacun d'entre eux.

3.2. La co-simulation multi-niveaux

Un modèle de simulation multi-niveaux est une spécification exécutable contenant un ensemble de modules décrits à différents niveaux d'abstraction.

Le problème qui se pose dans de tels modèles est de pouvoir connecter automatiquement et à moindre coût deux ou plusieurs modules communicants chacun en utilisant des types de données et des protocoles de communication différents. La figure 3 1 montre un exemple d'un système contenant deux modules : module 1 et module 2. Le premier est décrit au niveau UTF et communique des données de type entier, via trois ports en utilisant des FIFO bloquantes. Le second module communique des bits et vecteurs de bits via 7 ports, avec des protocoles (maître-esclave, rendez-vous,..etc).

Ainsi, le problème est de trouver la meilleure façon de convertir les types de données et les protocoles pour permettre la communication entre ces deux modules du système.

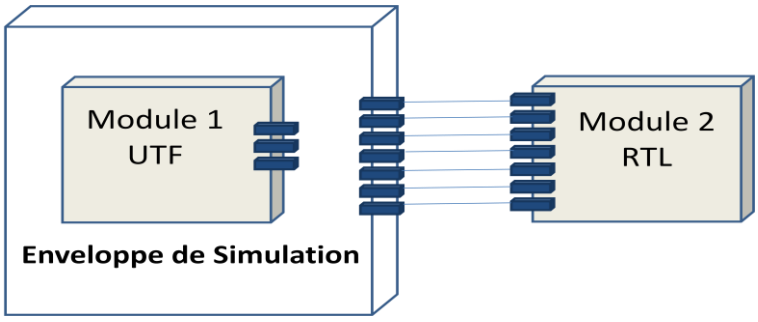


Figure 3.1 : Enveloppe de simulation multi-niveaux

3.2.1. Etat de l'art

Les travaux portants sur la co-simulation ont commencé à aborder l'approche multi-niveaux depuis une quinzaine d'années maintenant. Ce type de co-simulation implique l'exécution conjointe des composants décrits à différents niveaux d'abstraction.

Parmi ces solutions, le modèle fonctionnel de bus BFM [BrKr00] constitue une alternative intéressante pour l'interconnexion des modèles de simulation fonctionnels et des modèles de simulation au cycle prés, surtout pour la validation de l'interfaçage logiciel/matériel. Malheureusement cette méthodologie ne prend en compte que les accès mémoire, et en aucun cas elle ne permet d'adapter ou de transformer les primitives de communication de haut niveau.

Les travaux de G. Nicolescu et A. Jerraya dans l'outil ROSES [NIC02] permettent de construire des enveloppes de simulation, pour adapter les niveaux d'abstraction. Ces enveloppes générées à partir d'une bibliothèque de composants élémentaires. La difficulté de maintenir cette dernière à une taille raisonnable me semble être l'inconvénient de cette approche.

CoWareN2C [COW02] est un environnement qui offre une solution de co-simulation multi-niveaux. Les niveaux d'abstraction proposés sont : BCA qui est proche du niveau RTL

et le UT qui est un modèle fonctionnel ne prennent pas en compte le temps. Ainsi CoWareN2C présente un concept appelé BCASH, il s'agit d'une enveloppe des sous systèmes au niveau UT permettant d'estimer le temps nécessaire à l'exécution d'une procédure dans un tel modèle Cette enveloppe peut être générée automatiquement seulement dans le cas où le sous système UT est ciblé en logiciel (c'est-à-dire qu'il sera exécuté sur un simulateur de processeur (ISS)), ce qui est malheureusement une contrainte forte. Un outil plus récent de CoWare (CoWare® Platform Architect) [COW10] permet aujourd'hui, d'intégrer des blocs RTL dans des spécifications TLM. Ce que l'on peut, à mon sens, reprocher à cette méthodologie c'est le fait qu'elle soit très (trop) orientée vers les IPs propriétaires de CoWare. De plus, elle ne s'applique qu'à des descriptions TLM respectant le standard dans son état actuel.

D'autres environnements comme celui de Chinook [ChOr95] et [SeGh02] proposent des méthodologies permettant de changer dynamiquement pendant la simulation les niveaux d'abstraction des différents modèles. Cependant, ils semblent cibler des protocoles et niveaux bien précis.

On trouve également dans la littérature une autre classe d'approches, proposant des extensions de langages pour supporter la simulation multi-niveaux. C'est le cas dans SystemC^{SV} [SiMu01] et [BNR06]. Ces méthodes ont certes l'avantage de proposer des moteurs "optimisés" pour ce genre de simulation, mais leur utilisation suppose un changement d'habitude des concepteurs. Ceci est souvent loin d'être évident.

3.2.2. Méthodologie de simulation multi-niveaux

Notre contribution est la proposition d'une nouvelle approche de validation de systèmes monopuces par simulation. Avec cette approche, on peut réaliser une simulation rapide et peu coûteuse de systèmes construits par assemblage d'IPs. Ces derniers doivent être décrits en SystemC, mais peuvent être à différents niveaux d'abstraction.

L'originalité principale de ce travail est que la méthodologie présentée ne nécessite aucune bibliothèque externe à SystemC.

Partant de la description des interfaces des modules, constituant l'application, en SystemC ainsi que du niveau d'abstraction de la simulation voulu par le concepteur. Nous générons un modèle de simulation du système. Ce modèle est constitué des modules de départ et d'une enveloppe de simulation associée à chacun des modules décrits à un niveau d'abstraction différent de celui de la simulation. La génération du modèle exécutable de simulation se fait essentiellement en trois étapes (figure 3.1) : analyse du code décrivant les interfaces de communication des modules, construction des adaptateurs de simulation et la connexion des adaptateurs générés avec les modules initiaux.

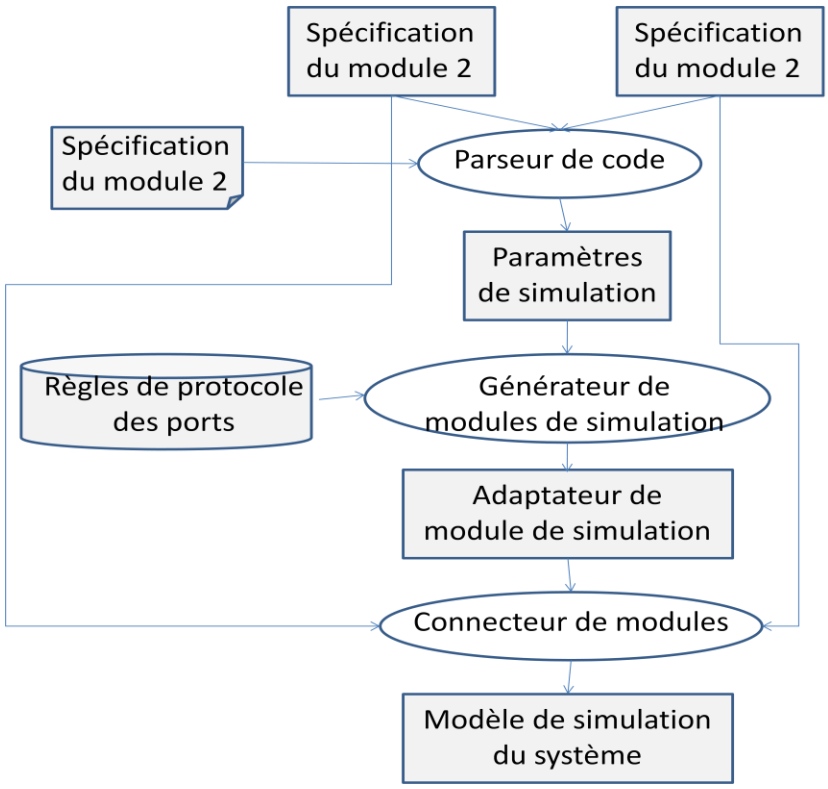


Figure 3.2. Flot de génération des enveloppes de simulation pour adaptation de niveaux

3.2.3. Génération des enveloppes de simulation

L'enveloppe de simulation de tout module est réalisée par l'instanciation d'une classe appelée *GenericInterface*. Cette classe est unique et configurable, ce qui rend la structure du modèle de simulation final relativement simple et sa génération facile.

Le processus SystemC infini décrivant le comportement de chaque enveloppe de simulation est généré en utilisant des règles de transformation de protocoles et d'association, préalablement définies par un concepteur, et pouvant être réutilisées. Ainsi ces règles sont principalement de deux types :

3.2.3.1. Règles d'association : Ce sont des règles simples décrivant l'association des ports entre des modules hétérogènes en termes de niveaux d'abstraction. Ainsi par exemple une mémoire décrite au niveau RTL communique via un certain nombre de ports contrôle, de paquet de ports d'adresse et des ports de données, un processeur décrit au niveau UTF accède à une mémoire simplement avec un port d'adresse et un autre de donnée tous deux de types entier, et éventuellement un ports de contrôle de la mémoire (WriteEnable par exemple).

3.2.3.2. Règles de transformation : Ce sont des règles toutes aussi simples que les précédentes mais qui définissent les éventuelles transformations de protocoles entre les modules. Ainsi dans l'exemple de la mémoire et du processeur, un changement de valeur sur le port de donnée du processeur implique l'écriture sur un ensemble de ports de la mémoire dans un ordre bien précis (protocole d'accès à la mémoire fourni par le constructeur de la mémoire).

3.2.4. Structures des enveloppes de simulation

Nos enveloppes de simulation sont composées essentiellement de trois parties: les ports logiques, les ports réels et un processus infini SystemC comme montré dans la figure 3.

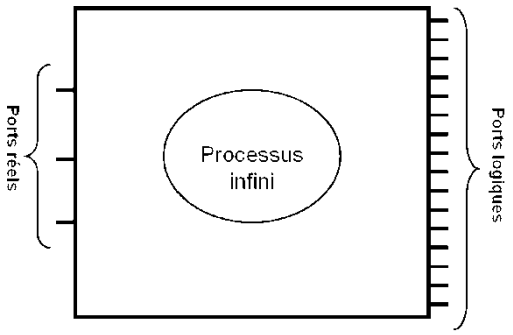


Figure. 3.3. Structure d'une enveloppe de simulation

3.2.4.1. Ports réels

Ils correspondent aux ports du module initial. Ils ont donc exactement les mêmes caractéristiques que ceux de ce dernier (type, données, protocoles). Ainsi, les ports réels de l'enveloppe de simulation de l'exemple de la figure 3.1 sont identiques à ceux du module 1.

3.2.4.2. Ports logiques

L'enveloppe de simulation doit posséder le même nombre de ports logiques que le module réel avec lequel elle sera connectée.

Ainsi, dans la figure 3.1, les ports logiques de l'enveloppe de simulation sont exactement similaires à ceux du module 2.

3.2.4.3. Processus infini

C'est un processus SystemC généré automatiquement en instanciant et en configurant une class C++ que nous avons implémenté. Il est sensible aux signaux connectés aux ports réels de l'enveloppe. Ainsi, il lit les données sur ces ports puis les convertit pour les adapter aux signaux connectés aux ports logiques de l'enveloppe.

3.2.5. Thèses et masters

Ces travaux sur la simulation multi-niveaux ont été réalisés principalement dans le cadre des masters [VEN03], [KOU04] que j'ai encadré à 100% et la thèse de Joel Vennin [MVD03a, MVD04] que j'ai codirigée à 50%.

3.2.6. Publication représentative

Titre
A Fast SystemC Simulation Methodology for Multi-level IP/SoC Design ,
Publication
International Workshop on IP Based SoC design 2003
Auteurs
Samy Meftali, Joel Vennin et Jean-Luc Dekeyser
Résumé
Nous présentons dans ce papier une méthode rapide permettant de connecter des modules décrits en SystemC. Ces modules peuvent être décrits à différents niveaux d'abstraction, et nous obtenons un modèle de simulation exécutable du système entier.
L'originalité de cette approche réside dans le fait qu'elle ne nécessite pas l'utilisation de bibliothèques externes à SystemC. En effet, le modèle de simulation généré est basé uniquement sur : les descriptions des différents modules composant le système, une bibliothèque de classes rajoutée à SystemC et d'un langage de description de règles.
Ainsi, avec notre méthodologie, nous générons des enveloppes de simulation pour les modules du système, d'une manière rapide et automatique, en instanciant une classe SystemC générique et configurable. Ces enveloppes sont produites d'une façon rapide et automatique.
L'efficacité de notre approche est montrée sur une application composée d'une SDRAM au niveau transfert de registres et d'un module transmetteur au niveau fonctionnel.

3.2.7. Analyse et discussion

A l'époque où ces travaux ont été réalisés, les pratiques répondues pour cette problématique pouvaient être classées principalement en deux classes : interfaces manuelles et construction des enveloppes en utilisant des bibliothèques de briques élémentaires. Le problème de la méthode manuelle réside dans le fait qu'elle se repose entièrement sur l'expérience du concepteur. La seconde approche décharge les concepteurs (partiellement) de cette tâche. Cependant, elle fait apparaitre un autre problème, tout aussi important, qui est la taille des bibliothèques utilisées.

J'ai donc proposé dans mes travaux une nouvelle approche en essayant d'éviter les inconvénients des méthodes existantes. Ainsi, les enveloppes de simulation que l'on génère sont basées sur description de règles textuelles et non sur des bibliothèques. La production du code est automatique, mais il reste tout de même à la charge du concepteur de spécifier manuellement les règles.

La tendance aujourd'hui est d'intégrer les méthodes permettant l'interopérabilité, sous forme d'APIs et bibliothèques, directement dans les langages de description. C'est le cas depuis l'intégration de la bibliothèque TLM à SystemC récemment. Les nombreuses méthodes et types fournis facilitent certes la tâche du concepteur, cependant, il lui appartient toujours d'effectuer (manuellement) toutes les instanciations et d'écrire le code assurant l'interopérabilité. Ainsi, il s'agit quand même d'une opération manuelle pouvant être fastidieuse et source d'erreurs.

Aujourd'hui, il me parait particulièrement intéressant de développer des méthodologies et outils permettant de générer automatiquement la couche assurant l'interopérabilité entre niveaux d'abstraction. Ceci serait sans doute pertinent avec le consensus qui se généralise autour des niveaux TLM. Ces méthodes d'automatisation pourraient réutiliser les API fournis avec la bibliothèque TLM de SystemC par exemple, et adopter une approche IDM pour génération automatique du code. L'utilisation de l'IDM dans un tel outil augmenterait, en plus, significativement les possibilités de réutilisation des modèles d'interopérabilité. Cela

permettra également de s'attaquer au problème de l'interopérabilité sans se lier à un langage de simulation particulier (comme SystemC). Une telle approche serait flexible et ouverte vers d'éventuels langages qui feraient leur apparition dans les années à venir.

3.3. La co-simulation hétérogène, géographiquement distribuée.

Un modèle de simulation distribuée est une spécification exécutable, sur une plateforme constituée de sites distants et connectés via un réseau. Elle contient un ensemble de modules (IPs) décrits en SystemC (il s'git d'un choix). Le problème qui se pose dans de tels modèles est de pouvoir connecter et synchroniser automatiquement les IPs distants afin de permettre leur communication.

La figure 3.4 montre un exemple d'un système composé de plusieurs IPs. Ce système est simulé sur une plateforme de simulation composée de quatre sites distants. Chacun de ses sites exécute un simulateur.

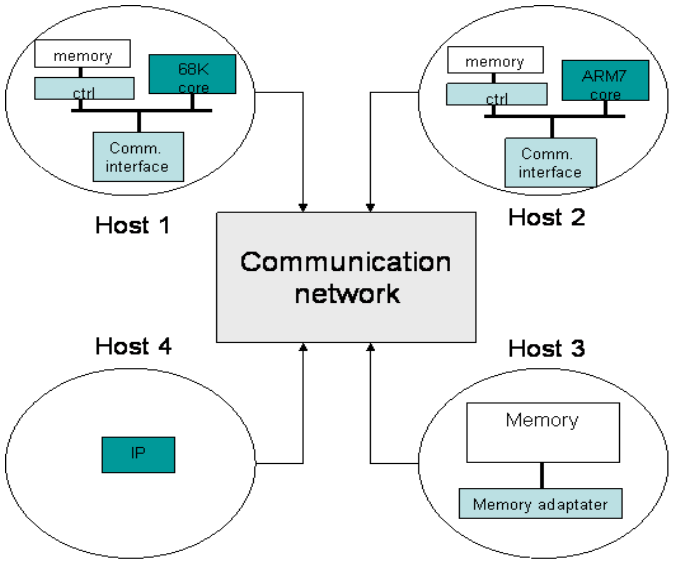


Figure 3.4. : Illustration d'une simulation distribuée sur quatre machines

3.3.1. Etat de l'art

Il ya eu plusieurs tentatives pour distribuer des simulations SystemC, mais il semble qu'aucune ne soit définitive aujourd'hui ou utilisée comme standard.

Il existe essentiellement deux approches pour distribuer des simulations SystemC. Certains auteurs ont essayé de paralléliser et distribuer SystemC en modifiant directement le moteur de simulation. L'autre stratégie est basée sur les communications entre les simulateurs en utilisant des bibliothèques de communication au dessus de la couche SystemC.

Un inconvénient majeur à la modification du moteur de simulation SystemC est la nécessité de fournir une maintenance continue pour s'adapter aux futures implémentations du langage. Bien que, d'autre part, il s'agisse d'une approche plus personnalisable.

[COX05] and [CCZ06] ont adopté la première stratégie en proposant de personnaliser le moteur de simulation SystemC. Les deux propositions utilisent le Message Passing Interface (MPI) [DHO98] comme moyen de communication entre les simulateurs, qui sont enveloppés dans un module de haut niveau SystemC. Cette stratégie obtient des résultats raisonnables pour les modèles de simulation bien équilibrés entre les processeurs et décris à gros grain. Cependant les modules doivent être distribués manuellement.

D'autres propositions suivant la deuxième approche, inclurent leurs propres bibliothèques de communication et synchronisation des modules. Ainsi [HAM05] présente une solution basée sur MPI, tandis que [TRA04], [RW] et [FH] proposent l'utilisation de la communication par sockets TCP/IP.

Enfin, on trouve également des méthodes de distribution de la simulation basées sur des protocoles internet ou des middlewares tels que SOAP ou RMI, comme dans [MDC05].

Il semble aussi que dans tous ces outils, l'affectation des modules, décrivant le système, aux différents simulateurs distants soit basée entièrement sur l'expérience du concepteur sans aucune optimisation rigoureuse. Ceci peut s'avérer être un handicap considérable si les volumes et les fréquences de communication inter modules sont importants.

3.3.2. Méthodologie de co-simulation distribuée

Notre méthodologie de génération de modèles de co-simulation distribués est principalement composée de deux étapes. Ainsi, à partir de la description des interfaces de communication, des modules SystemC, composant le système, on trouve une affectation optimale de ces modules sur les hôtes de simulation disponibles. Puis nous transformons le code pour adapter le modèle de simulation au contexte de l'exécution sur un réseau distribué.

La première étape est basée sur une méthode d'optimisation combinatoire. Elle consiste à générer une affectation optimale des modules simulés sur les différents hôtes de simulation disponibles. La seconde étape consiste à transformer le code initial de l'application, afin de l'adapter à la plate-forme de simulation distribuée, trouvée lors de l'étape précédente.

3.3.2.1. Optimisation de la simulation distribuée

Ma contribution sur ce point consiste en la proposition d'une nouvelle méthodologie permettant une distribution optimisée des IPs SystemC constituant un SoC sur simulateurs distribués et géographiquement distants. La principale originalité de ce travail est l'utilisation d'une formulation mathématique exacte pour résoudre le problème de distribution d'IPs. Ainsi, cela peut accélérer considérablement la simulation et par conséquent réduit le temps de mise sur le marché.

A partir de la spécification de SystemC de chacun des modules constituant le système, un parseur analyse le code pour extraire certains paramètres liés à l'application. Ces paramètres sont essentiellement les caractéristiques des signaux échangés entre les modules (types et tailles des signaux). Ensuite, une simulation rapide du système sur un seul hôte permet d'extraire d'autres informations comme la quantité de données échangées entre chaque paire de modules et la charge CPU nécessaire pour exécuter chaque module.

Après cette étape d'extraction de paramètres, le flot a besoin de certaines autres informations du concepteur. Ce sont principalement: la vitesse de transmission et de la bande passante entre toutes les paires de machines hôtes connectées via le réseau.

Ces paramètres et informations permettent d'instancier un programme linaire en nombres entiers. Ce dernier est résolu automatiquement et de façon transparente pour le concepteur. Il fournit en sortie une affectation optimale des différents modules composant le système sur les machines de simulation. Cette affectation constitue l'entrée d'un second flot, permettant d'adapter automatiquement les codes des IPs de simulation pour prendre en compte les hôtes qui leurs sont attribués. Cette transformation de code permet de rajouter toute la couche de communication réseau, de façon à ce que le concepteur se comporte exactement comme si la simulation était sur une seule machine locale.

La formulation mathématique du problème d'affectation, ainsi que le flot permettant l'instanciation et la résolution du modèle sont décrits avec détails dans [MeDe04c, MVD06].

3.3.2.2. Communication via le réseau

La seconde étape de notre flot de génération de modèles de simulation distribués est la production du code permettant la communication des différents modules de simulation, affectés aux machines distribuées. En effet, la co-simulation dans un environnement constitué de plusieurs hôtes implique une synchronisation des différents simulateurs.

Voulant répondre en même temps au problème de l'hétérogénéité, en termes de langages, dans les spécifications de SoCs complexes, je me suis imposé dans mes travaux la contrainte de rendre l'environnement de simulation assez flexible pour supporter des descriptions multi-langages.

3.3.2.2.1. SystemC et des réseaux

La norme API SystemC ne permet pas les simulations distribuées sur un réseau. Ainsi, afin de rendre possible cette fonctionnalité, nous avons défini deux nouveaux concepts: enveloppes de distribution et objets de transport.

3.3.2.2.2. Pourquoi utiliser CORBA ?

CORBA (Common Object Request Broker Architecture), est une solution proposée par l'OMG (Object Management Group) pour résoudre le problème d'interopérabilité dans le cas de logiciels multi langages distribuées. Ainsi, l'OMG spécifie le standard de description d'interfaces (IDL) et les APIs permettant l'interaction entre les objets distribués en utilisant l'ORB (bus CORBA).

L'ORB assure l'interconnexion des applications situées sur des machines distantes, dans un environnement hétérogène distribué. Dans une application traditionnelle de type Client/Serveur, le programmeur doit définir son propre protocole de communication entre les objets composant son application. Le rôle de l'ORB est de simplifier cette tâche, en rendant la couche protocole complètement transparente pour le développeur. Ainsi, le protocole est défini par l'interface de l'application en utilisant un seul langage de spécification, indépendant de la mise en œuvre (IDL).

L'utilisation de CORBA est donc ainsi justifiée. En plus, il fournit plusieurs services par défaut, comme les services de nommage, d'authentification, etc. L'ensemble des services proposés peuvent être très pratiques pour mettre des IPs, à disposition des concepteurs, sur des machines distantes.

3.3.2.2.3. Comment cela fonctionne ?

L'idée principale de ce travail est de permettre aux concepteurs d'utiliser des IPs, disponibles dans des hôtes éloignés, appartenant à différents fournisseurs. Donc, nous devons être capables de simuler un système conçu en utilisant des IPs distribués. En effet, Il arrive assez souvent, lors de la conception d'un SoC complexe, que le concepteur ait besoin d'utiliser certains IPs répartis sur plusieurs sites distants (pour des raisons de propriété intellectuelle notamment). Ainsi, notre solution est capable de simuler un système utilisant plusieurs simulateurs SystemC, s'exécutant sur des machines distantes, communiquant en utilisant l'ORB.

Le principe consiste à intégrer au système à concevoir des IPs lointains. Ainsi, l'utilisation du service de nommage CORBA permet, à l'outil, de localiser un IP distant en utilisant simplement son nom ou sa description. Ce mécanisme est complètement transparent pour le concepteur.

Toute la couche relative à la communication réseau entre les simulateurs est générée de façon totalement automatique, à partir des spécifications initiales des IPs à simuler. Les détails de cette approche sont décrits dans [MeDe04b, MVD03b].

3.3.3. Thèses et master sur cette problématique

Les travaux sur la simulation distribuée ont été réalisés essentiellement dans le cadre du travail de thèse de Joel Vennin [MeDe04a, MeDe04b, MVD03b, MVD06] que j'ai codirigé à 50% et le master [TUR04] encadré à 50%.

3.3.4. Publication représentative

Titre
Automatic Generation of, Geographically Distributed, SystemC Simulation Models for IP/SoC Design
Publication
46th IEEE International MWSCAS, Cairo, Egypt, December 2003
Auteurs
Samy Meftali, Joel Vennin and Jean-Luc Dekeyser
Résumé
Dans cet article, nous présentons une nouvelle méthodologie permettant la génération automatique de modèles de simulation SystemC, géographiquement distribués, pour la conception de SoC.s
Nous construisons les modèles de simulation SystemC comme des applications client/serveur, où chaque client/serveur contient quelques IPs composant le système. Le nombre de simulateurs SystemC distribués pouvant être utilisés dans la plateforme de simulation est illimité. La communication entre eux est faite en utilisant le bus CORBA.
L'efficacité et la simplicité de notre approche sont illustrées sur un exemple d'un système composé de trois IPs distribués.

3.3.5. Analyse et discussion

Le concept de la simulation distribuée me semble très important au regard de la complexité des systèmes embarqués modernes, et de la façon avec laquelle ils sont conçus. En effet, trois constats me laissent penser, aujourd'hui, que la simulation distribuée restera l'un des défis majeurs dans les prochaines années.

- Les systèmes sont hétérogènes : ils intègrent des parties électroniques, mécaniques, optiques, etc.
- Les équipes d'ingénieurs sont spécialisées : plusieurs équipes participent à la conception des systèmes complexes. Chacune développe certaines parties, selon son expertise.
- Réutilisation intensive des composants : il serait complètement aberrant aujourd'hui d'imaginer de concevoir un système embarqué complexe à partir de zéro (from scrach). Je suis même convaincu que la réutilisation d'IP (provenant de fournisseurs divers) ne fera qu'augmenter dans le futur.

On peut également ajouter à ces éléments l'élévation des niveaux d'abstraction et leur standardisation (TLM). Ceci permet en effet d'échanger (via le réseau) des transactions complexes au lieu des bits (comme c'est le cas dans les descriptions de niveau RTL).

Ces même constats, font qu'un certain nombre de plateformes virtuelles telles que OpenPeople font leur apparition aujourd'hui. Elles mettent à disposition des concepteurs des modèles de simulation que l'on peut instancier à distance, pouvant éventuellement cohabiter avec des IPs locaux dans un même projet de conception. Ainsi, il ne semble pas exclu que l'on s'oriente dans les années à venir vers des plateformes de simulation de type P2P. Ce

genre d'outil tout à fait envisageable avec l'apparition récente de standards d'intégration tels que IP Xact.

Les défis futurs relatifs à la simulation distribuée, concerneront à mon sens, particulièrement deux aspects :

- Standardisation de l'interopérabilité entre simulateurs : c'est sans doute le rêve de tout concepteur de pouvoir faire interagir plusieurs simulateurs en s'affranchissant des détails techniques liés à leur connexion et leur communication, et ce quelque soit leurs fournisseurs.
- Optimisation de l'environnement et minimisation du temps de communication via le réseau : le problème majeur de la simulation distribuée est le délai d'échange via le réseau. Des méthodes optimisant la distribution des IP sur les machines de simulation disponibles (quand cela est possible) et réduisant les échanges entre simulateurs pourront faire profiter les concepteurs des avantages de la simulation distribuée tout en réduisant ses inconvénients majeurs.

3.4. L'estimation de la consommation des systèmes sur puce

La description d'un composant au niveau CABA est une description précise au cycle près. De ce fait, estimer la consommation d'un système au niveau CABA permet de déterminer la configuration la plus adéquate de la microarchitecture de ses composants. En outre de la précision que ce niveau offre, la simulation s'exécute en un temps réduit par rapport au niveau RTL et les niveaux les plus bas, ce qui représente un gain de temps permettant de réduire le temps de mise sur le marché.

Pour pouvoir estimer la consommation d'énergie des différents composants d'une architecture, il faut tout d'abord comprendre les principes de leur description au niveau CABA. A ce niveau d'abstraction, le comportement du système est décrit à chaque cycle. Chaque composant est décrit par une ou plusieurs machines d'états communicantes synchrones (Synchronous Communicating Finite State Machines) [13]. La communication entre les FSM (Finite State Machines) est réalisée en partageant des signaux qui permettent l'échange des données.

Les composants dont nous estimons la consommation dans ce projet font partie de la bibliothèque SoCLib [SOC06]. C'est une bibliothèque de composants matériels qui permet de réaliser un simulateur de MPSoC au niveau CABA. La description des FSM de certains composants de cette bibliothèque est détaillée dans la thèse de Ben Atitallah [BEN08].

Dans ces FSMs, à chaque cycle une transition est réalisée, ce qui correspond à une consommation d'énergie nécessaire pour une activité du composant telle que la lecture, l'écriture et l'attente pour le composant mémoire. Ces consommations élémentaires peuvent être estimées pendant la simulation de diverses manières. Ainsi, l'objectif de mes travaux dans cette thématique est de proposer des méthodes permettant de réaliser des estimations de la consommation, pendant la simulation, de façon précise et rapide

3.4.1.La consommation statique et la consommation dynamique

La consommation d'énergie pour un circuit est celle liée à la transition des nœuds logiques (consommation dynamique) et celle correspondant à l'état de repos du circuit (consommation statique). Ce deuxième type de consommation a été négligé pour longtemps pour des technologies où les courants de fuites étaient faibles. Mais, aujourd'hui avec des technologies submicroniques (au dessus de 0,25µm), les courants de fuites sont devenus de plus en plus importants. De ce fait, la consommation statique devient un élément important dans l'estimation de la consommation totale du circuit.

3.4.2.Les outils d'estimation de la consommation

Il existe un grand nombre de méthodes et outils, pour l'estimation de la consommation des SoCs, dans la littérature. Les niveaux d'abstraction auxquels ces outils opèrent, constituent une des façons de les différencier et les classifier. Ainsi, je présente dans la suite de cette section quelques outils connus pour chaque niveau d'abstraction, en allant du niveau transistors au niveau fonctionnel.

3.4.2.1. Niveau transistors

L'estimation de la consommation à ce niveau se base sur les courants électriques qui circulent dans les transistors, ce qui nécessite une description du SoC au niveau transistor. Parmi les outils opérant à ce niveau, nous trouvons SPICE [SPA], Lsim Power Analyst de Mentor Graphics [LSI04] et Power Mill de Synopsis [POW].

L'estimation à ce niveau a l'avantage de donner des résultats très proches des valeurs réelles mais elle a l'inconvénient d'un temps de simulation important. Cette lenteur constitue un obstacle pour l'estimation de la consommation d'une architecture à haut degré d'intégration.

3.4.2.2. Niveau portes logiques

A ce niveau, l'estimation de la consommation se base sur une bibliothèque de portes logiques permettant l'association d'un coût de consommation à chaque type de porte logique et l'utilisation de ces données pour déterminer la consommation du circuit. Parmi les outils utilisant cette technique, nous trouvons PowerGate de Synopsys [PoGa].

3.4.2.3. Niveau RTL

Ce niveau d'estimation se base sur la description de l'implémentation sous forme de registres et de bascules. Parmi les outils qui utilisent cette approche, nous trouvons Petrol de Philips [LlGo98].

L'estimation de la consommation aux trois niveaux présentés ci-dessus souffre d'une lenteur considérable, ce qui rend difficile l'exploration des différentes solutions architecturales. De plus, ces trois niveaux ne permettent pas d'étudier l'évolution de la consommation au cours de l'exécution de l'application du fait qu'ils se basent uniquement sur la partie matérielle.

3.4.2.4. Niveau architectural

Ce niveau d'estimation se base sur des niveaux d'abstraction plus élevés. Ce niveau estime la consommation correspondante à des évènements qui ne concernent pas la commutation dans un transistor ou le changement d'état d'une porte logique, mais qui sont liés aux activités des composants telles que l'exécution d'une instruction par un processeur ou l'accès mémoire.

L'analyse de la consommation par trace : La première approche est proposée par l'outil AVALANCHE [HEN99]. Dans cette approche, l'application est exécutée en utilisant un simulateur de processeur décrit au niveau du jeu d'instruction (Instruction Set Simulator ISS). La trace de l'exécution de l'application sert à estimer la consommation du processeur. Pour la consommation de la hiérarchie mémoire, il y a aussi une trace des accès fournie par un traceur de requêtes. Cette trace est traitée par l'outil Dinero [LiHe98], pour en détecter les succès et les défauts de cache et les accès aux mémoires de données ou d'instructions. Les occurrences des activités du processeur et de la hiérarchie mémoire sont transmises à des modèles d'énergie, selon le type du composant, pour calculer la consommation totale. L'inconvénient de cette approche est qu'elle ne considère pas la consommation de l'interconnexion ni celle des interactions temporelles entre les composants telles que les délais dus aux conflits lors des accès simultanés à une mémoire.

L'analyse de la consommation au cycle près : Cette approche vient remédier à l'imprécision de l'approche précédente en se basant sur une simulation au cycle près. Elle assure la synchronisation entre les différents composants du système décrits avec un langage de description comme le C et le SystemC. Un modèle de consommation pour chaque composant est intégré dans le simulateur pour permettre le calcul de la consommation à chaque cycle en se basant sur les occurrences des activités pertinentes des composants. Parmi les outils utilisant cette approche, on peut citer Wattch [BTM00] et SimplePower [YVK00]. L'inconvénient de cet outil est qu'il ne permet pas l'estimation de la consommation d'une architecture multiprocesseur (MPSoC) mais se base sur un seul processeur super-scalaire.

3.4.2.5. Niveau fonctionnel ou algorithmique

Ce niveau d'estimation étudie la consommation d'un programme. La première méthode pour ce type de consommation est la méthode de Tiwari et al. [TMW94]. Cette approche consiste à cumuler l'énergie consommée par chaque instruction pour trouver la consommation totale du programme. Cette méthode nécessite beaucoup d'analyses et de mesures et par conséquent un temps important pour développer le bon modèle de consommation pour un processeur cible.

Parmi les outils qui utilisent cette méthode, nous citons l'outil JouleTrack [SCJ00], développé par Sinha et al., qui permet d'estimer la consommation pour des processeurs simples tels que les RISC.

L'outil SoftExplorer [SLJ04], développé LabSticc, vise à étendre la méthodologie de Tiwari pour des processeurs plus complexes et à réduire le temps de développement des modèles de consommation. Cet outil utilise une méthode qui s'appelle FLPA (Functional Level Power Analysis) [LJS04]. Elle se base sur un nombre réduit de mesures permettant de déterminer des paramètres algorithmiques, agissant sur la consommation, tels que le taux du parallélisme et le taux d'occupation des unités de traitement. Les lois de la consommation sont extraites en exécutant des applications élémentaires sur le processeur. Ces lois s'écrivent en fonction des paramètres algorithmiques et des paramètres de la configuration tels que la fréquence et le placement des données en mémoire. Après avoir déterminé ces lois, elles peuvent être appliquées à chaque application pour déterminer sa consommation totale. Ainsi, une analyse du code est faite pour extraire les paramètres algorithmiques. Ces paramètres sont insérés par la suite dans les équations du modèle de consommation de l'application. L'avantage de cette méthode est sans doute son temps d'estimation de l'ordre de quelques secondes.

3.4.3. Approche hybride pour l'estimation de la consommation des MPSoCs au niveau CABA

La consommation d'énergie de tout composant matériel est due à ses activités élémentaires. Notre méthodologie d'estimation de la consommation au niveau CABA consiste à associer à chaque activité un coût énergétique qui représente la consommation du composant à un cycle donné. Puis en accumulant les consommations de chaque cycle durant la simulation, nous pouvons déterminer la consommation totale. Cette méthodologie est illustrée par la figure 3.3. Cette figure représente la FSM d'un composant x qui a quatre états S_0, S_1, S_2 et S_3. S_0 est l'état initial. Les flèches discontinues montrent l'évolution de cette FSM au cours de la simulation. A chaque cycle, nous pouvons déterminer par cumulation des consommations, la consommation totale du composant x depuis le début de la simulation.

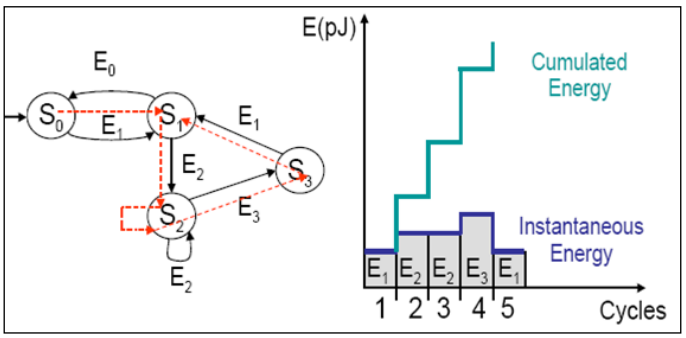


Figure 3.3 Estimation de la consommation d'un composant à partir de sa FSM

Pour estimer la consommation de chaque composant du MPSoC, il faut déterminer son modèle de consommation et intégrer ce modèle dans le simulateur qui prend en compte les paramètres architecturaux et technologiques.

L'estimation de la consommation au cours de la simulation permet de déterminer quel est le composant qui consomme le plus d'énergie et permet ainsi au concepteur de réduire l'énergie consommée en agissant sur les paramètres architecturaux et technologiques tels que la taille du cache ou en adoptant une autre alternative architecturale. Ainsi, en testant plusieurs alternatives, nous pouvons déterminer celle qui a la consommation la plus acceptable.

Une estimation suffisamment précise de la consommation revient à déterminer les activités effectuées à chaque transition et le coût énergétique de chaque activité. Ainsi pour déterminer la consommation totale d'un composant, il suffit de faire la somme des consommations des différentes activités. La figure 3.4 [BEN08] illustre cette approche dans l'estimation de la consommation.

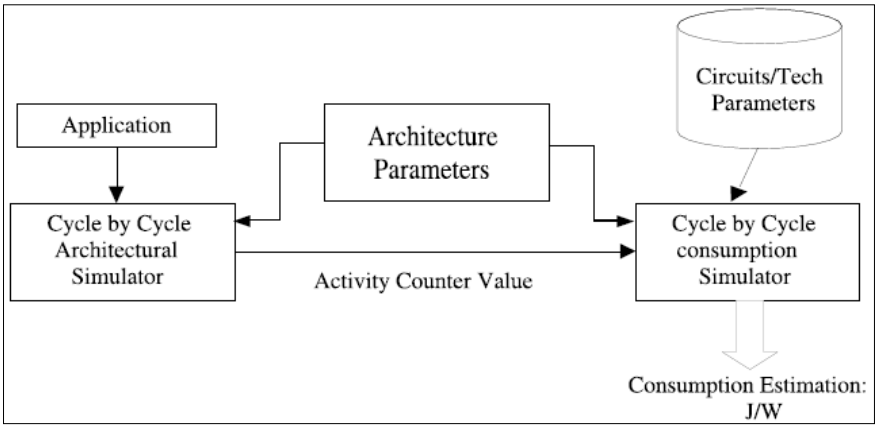


Figure 3.4 Estimation de la consommation au niveau CABA

Cette figure montre que la simulation au niveau CABA permet de donner les compteurs des activités réalisées par les différents composants. Ces compteurs servent comme entrées pour les différents modèles de consommation qui prennent en compte les paramètres architecturaux et technologiques pour déterminer la consommation totale de chaque composant et celle de l'architecture complète.

Cette approche est basée donc sur deux étapes :

1. La localisation des activités pertinentes en terme de consommation d'énergie et associer à chaque activité un compteur afin de connaître le nombre d'occurrences correspondant lors de la simulation.

2. L'évaluation du coût énergétique élémentaire de chaque activité pertinente. Celui-ci dépend des paramètres architecturaux et technologiques.

Les consommations relatives à chaque activité pertinentes d'un composant ont été mesurées, pour la plupart, sur les circuits réels. Cependant, certaines ont été estimées avec des modèles mathématiques. Ces modèles et mesures sont expliqués de façon détaillée dans [BNGM06].

3.4.4. Approche d'estimation par espionnage

L'inconvénient de l'approche présentée dans la section précédente est que les occurrences des activités pertinentes sont extraites à partir des compteurs insérés dans le code des IPs. Ceci nécessite d'entrer dans le code de chaque composant et y insérer les compteurs d'activités à différents endroits. Ceci peut s'avérer très fastidieux, particulièrement si l'on n'est pas familier avec le code d'un IP. De plus, en général, un IP fini n'est pas modifiable et toute modification a pour conséquence de créer une nouvelle version, un nouvel IP.

Dans notre seconde approche, nous nous sommes inspirés du même principe, basé sur les consommations des différentes activités élémentaires. Cependant, on s'impose comme contrainte de ne pas être intrusifs dans les descriptions des IPs. La question qui se pose ici est comment récupérer ces machines d'états sans entrer dans le code des IPs ?

La solution à ce problème est la détection des activités des composants à partir des signaux qu'ils échangent et construire des FSM très proches de celles des composants simulés. Pour ce faire, des modules d'estimation de consommation seront connectés entre les composants au cours de la simulation. Ces modules détectent les requêtes et les réponses entre les composants et en dégagent la consommation d'énergie correspondante à chaque activité de ceux-ci. La figure 3.5 illustre cette approche en montrant deux espions de simulation intercalés entre deux modules A et B.

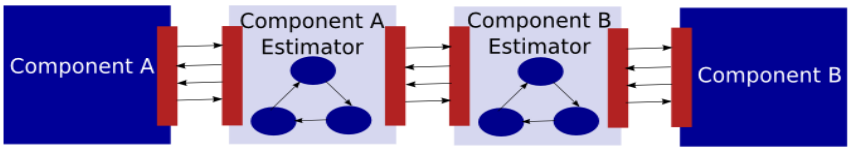


Figure 3.5 Estimation de la consommation par espionnage

3.4.4.1. Les protocoles de communication

L'estimation de la consommation par espionnage se base sur l'observation et l'analyse de signaux échangés entre les composants matériels. Une telle approche n'est réalisable que si l'on se conforme à un standard de communication bien défini.

Il existe plusieurs standards de protocoles (interfaces de communication) dans la littérature. CoreConnect [COR], VCI [VCI01], ARM [AMB] et OCP [OCP08] en sont des exemples bien connus.

J'ai choisi d'adopter le protocole OCP dans l'approche de simulation par espionnage pour tout un ensemble d'avantages qu'il offre comparé aux autres standards d'interfaces. Ces avantages sont principalement liés à sa grande flexibilité et sa richesse. Ils sont détaillés dans [TMB10].

3.4.4.2. L'estimation de la consommation d'énergie au niveau CABA

Dans cette approche, les coûts énergétiques élémentaires utilisés sont ceux obtenus par les expériences réalisées dans la thèse de Ben Atitallah [BEN08]. En fait, ces expériences se sont basées sur des simulateurs de bas niveau pour une technologie de fabrication de 90nm. A l'issu de ces expériences, des modèles de consommation qui dépendent d'un nombre réduit de paramètres ont été développés pour différents composants.

Nous nous sommes inspirés de ces modèles de consommation pour en produire d'autres en prenant en compte le fait que nous ne pouvons pas être intrusifs dans le code des IPs pour dégager les activités pertinentes mais nous les détectons à partir des signaux échangés entre les composants.

Ceci peut entraîner une erreur par rapport aux valeurs des consommations obtenues par la méthode de l'insertion des compteurs dans les IPs. C'est le cas quand la consommation ne peut pas être détectée par les signaux comme par exemple la consommation correspondante à l'écriture des requêtes en attente dans des FIFOs pour un cache ou une mémoire. Les expériences effectuées montrent que ce genre d'activités ne consomme pas une quantité importante d'énergie, ce qui permet d'obtenir une précision acceptable de l'estimation de la consommation même en les négligeant.

Notre approche de l'estimation de la consommation se base sur la construction d'une FSM pour chaque composant à partir des activités que nous pouvons détecter à l'aide des signaux échangés entre les composants. Ainsi, de telles FSMs ont été développées pour chaque de type principal de composants matériels comme : processeurs, mémoires, caches, bus, etc.

En plus de permettre l'interception des requêtes et leur interprétation, les interfaces OCP facilitent l'automatisation de l'approche d'estimation de la consommation. En effet, en supposant que tous les IPs des modèles de simulation soient conformes à OCP, un méta-modèle a été défini pour permettre la génération automatique d'espions dédiés aux différents types de composants matériels.

Remarque : l'hypothèse que tous les IPs soient conformes à OCP n'est pas forcément très limitative. En effet, l'adaptation d'autres protocoles vers OCP est une opération assez simple à réaliser.

3.4.5. Thèses et masters

Les travaux sur l'estimation de la consommation au niveau CABA en utilisant l'approche hybride ont été réalisés dans le cadre de la thèse de Rabie Ben Atitallah [BEN08], que j'ai co-encadré à 20%. Quant à l'approche par espionnage, elle a été définie dans le stage de master de Chiraz Trabelsi [TRA09], encadré à 100%.

3.4.6. Publication représentative

3.4.6.1. Approche hybride

Titre
An MPSoC performance estimation framework using transaction level modeling
Publication
IEEE RTCSA'2007, Daegu, Korea, 2007
Auteurs
R. Ben Atitallah, S. Niar, S. Meftali, J.-L. Dekeyser.
Résumé
Pour utiliser les ressources matérielles importantes disponibles dans la prochaine génération de systèmes multiprocesseurs sur puce (MPSoC) de manière efficace, des méthodes d'exploration d'architectures (DSE) rapides et précises sont nécessaires.
Dans cet article, nous présentons un environnement permettant des simulations rapides et l'évaluation des performances des MPSoC tôt dans le flot de conception, réduisant ainsi le temps de mise sur le marché.
Dans cet environnement et suivant l'approche de modélisation TLM, nous présentons une nouvelle définition du niveau PVT par l'introduction de deux sous-niveaux de modélisation complémentaires. Le premier, PVT Transaction Accurate (PVT-TA), offre un grand facteur d'accélération de la simulation par rapport au niveau CABA la modélisation de niveau. Le second, PVT événement précis (PVT-EA), offre une meilleure précision avec un facteur d'accélération toujours acceptable.
Une plate-forme MPSoC a été développée en utilisant ces deux sous-niveaux et intégrant des modèles d'estimation de performances. Les résultats de simulation montrent que la combinaison de ces deux sous-niveaux donne un bon facteur d'accélération de la simulation, pouvant aller jusqu'à 18 avec une marge d'erreur négligeable lors de l'estimation de performances.

3.4.6.2. Approche par espionnage

Titre
An MDE Approach for Energy Consumption Estimation in MPSoC Design
Publication
Rapido 2010
Auteurs
Chiraz Trabelsi, Samy Meftali, Rabie Ben-Atitallah, Jean-Luc Dekeyser, Smail Niar,
Résumé
La consommation d'énergie est un critère important à prendre en compte dans la conception des systèmes multiprocesseurs sur puce (MPSoC). Dans cet article, nous présentons une solution pour estimer les la consommation d'énergie tôt dans le flot de conception des MPSoC afin de trouver un bon compromis performances/consommation. Cette solution est basée sur l'insertion d'estimateurs de la consommation entre les composants matériels au cours de la co-simulation d'un système au niveau CABA (Cycle Accurate Bit Accurate).
Le code SystemC correspondant à ces estimateurs est généré automatiquement en utilisant une approche IDM. Notre solution offre un environnement d'estimation de l'énergie sans changer le code des IPs (propriété intellectuelle), ce qui permet leur réutilisation. La précision de cette approche est vérifiée par l'intégration l'estimation de la consommation lors de la simulation d'applications significatives.

3.7. Analyse et discussion

L'estimation de la consommation assez tôt dans la phase de conception permet un gain de temps considérable ainsi qu'une meilleure exploration de l'espace de solutions. La simulation au niveau CABA du fait de sa précision dans la description du comportement des composants de l'architecture, permet de dégager les activités pertinentes en termes de consommation de l'énergie de façon précise.

Cette précision dans la détermination des activités pertinentes est appuyée par une précision dans la détermination des coûts énergétiques associés à ces activités grâce à une approche hybride qui se base sur les résultats des simulations à bas niveau et sur des modèles analytiques. De ce fait, nous pouvons obtenir presque les mêmes résultats d'un simulateur de bas niveau dans un temps réduit ce qui est l'avantage d'une simulation à un niveau plus élevé.

Ce genre d'approches n'exclut certes pas les mesures physiques de consommation et de performances. Cependant, à l'aire des manyprocesseurs et le retour en force des SIMD dans le domaine de l'embarqué, il me semble bien clair qu'il serait inenvisageable, dans le futur, de se passer des méthodologies d'estimation de haut niveau. La dernière méthode présentée dans ce chapitre me semble aller dans le bon sens. En effet, elle est basée sur trois critères, qui à mon avis, sont de plus en plus importants (particulièrement dans la conception de systèmes complexes) :

- non intrusivité dans le code (séparation du code d'estimation de celui des IPs)
- automatisation (utilisation du standard OCP et l'approche IDM)
- rapidité de la simulation (niveau CABA)

Dans les prochaines années, les efforts en recherche sur cette problématique devraient se baser globalement sur ces mêmes critères. Cependant, il reste encore beaucoup à faire en termes de précision et de vitesse d'exécution. Les approches futures devraient également intégrer la consommation statique, car son poids est de plus en plus important, à cause de la finesse de gravure.

L'application de cette approche par espionnage me parait également être une solution intéressante pour le contrôle et la gestion de la reconfiguration dynamique. En effet, un contrôleur de reconfiguration distribué sur différents composants d'un FPGAs, rendrait la

structure de contrôle beaucoup moins complexe. L'observation et la prise de la décision de reconfiguration de manière collaborative pourrait également réduire les temps d'exécution. Il s'agit là d'une problématique que j'ai lancée en fin 2009 dans le cadre de la thèse de Chiraz Trablesi.

Chapitre 4: Outillage et intégration dans Gaspard

4.1. Introduction

Les SoCs complexes doivent certainement leur existence aux outils d'aide à la conception. En effet, ce sont ces derniers qui ont rendu possible la manipulation de systèmes composés de centaines de millions de transistors. Cependant, l'explosion exponentielle de la complexité des SoCs, de nos jours, pousse les concepteurs vers une quête incessante de nouveaux outils permettant de suivre l'évolution des technologies d'intégration.

Ainsi, pour réduire les coûts de développement et augmenter l'évolutivité, l'Ingénierie Dirigée par les Modèles (IDM) représente une véritable alternative. Cette approche s'appuie principalement sur le langage UML et sur l'initiative MDA (Model-Driven Architecture) dont le principe consiste en l'élaboration de modèles indépendants de toutes plates-formes et leur spécialisation via des transformations pour l'implémentation effective des systèmes (OMG, 2003).

D'autre part, les outils de conception de SoCs doivent impérativement s'appuyer sur une implémentation modulaire et flexible, basée sur des composants génériques et réutilisables. C'est effectivement dans ce cadre que les environnements, de CAD modernes, doivent être développés. En plus des possibilités de modélisation de très haut niveau, ces outils doivent permettre la génération systématique et automatique de code fiable, pour les besoins de simulation et d'implémentation sur des circuits.

Je me suis donc naturellement intéressé à la conception des outils de CAD dans mes travaux de recherche, en orientant mes efforts sur deux aspects qui sont, à mon sens, d'une importance capitale dans tout flot de conception. Le premier aspect et **la génération automatique de code SystemC** pour des simulations fonctionnelles et le second concerne **la production de code VHDL RTL pour l'implémentation sur FPGAs**.

4.2. L'environnement GASPARD2

Gaspard2 (*Graphical Array Specification for Parallel and Distributed Computing version 2*) [GAS] est un outil de développement de systèmes embarqués développé par l'équipe DaRT. L'objectif principal de Gaspard est de fournir un unique environnement de développement valable tout au long du processus de conception d'un système embarqué.

En effet, Gaspard doit permettre, à partir d'une description UML du système, de modéliser le système, le simuler, le tester et générer du code pour l'application embarquée et l'architecture matérielle.

Gaspard part d'une description générale du système en UML, puis par raffinement et transformation de modèles successifs génère le code de l'application et de l'architecture matérielle. La figure 4.1 présente l'environnement de conception Gaspard2.

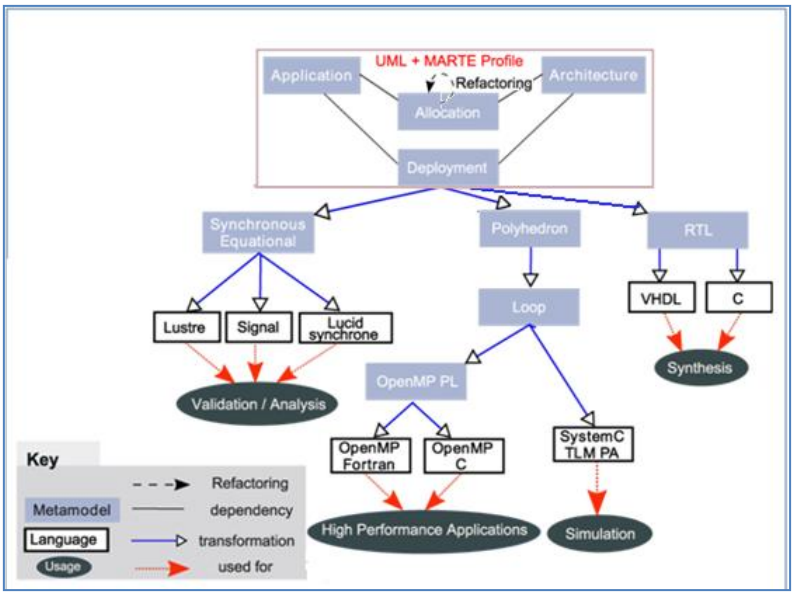


Figure 4.1. : L'environnement de conception de Gaspard2

Gaspard2 utilise le profile MARTE pour la modélisation de l'application, l'architecture et l'association de façon indépendante de la plateforme d'exécution. Par la suite, le profile de déploiement développé par l'équipe est utilisé pour ajouter les données de déploiement. A partir de cette modélisation, un ensemble de transformations de modèles est enchaîné pour obtenir le code dans un langage cible.

4.3. Les transformations automatiques modèle à modèle

Dans l'environnement Gaspard2, mes contributions ont comme objectifs de générer du code RTL pour cibler des FPGAs dynamiquement reconfigurables, en partant d'un modèle d'application en UML. Ainsi, trois transformations automatiques ont été développées et intégrées au cours de mes travaux à l'outil Gaspard2. Il s'agit de : *UML2MARTE*, *MARTE2RTL* et *RTL2CODE*.

4.3.1. Transformation *UML2MARTE*.

Pour la modélisation d'un automate de mode, nous utilisons les machines d'état d'UML dans l'environnement Gaspard2. Bien que ces concepts soient présents dans le modèle UML, elles ne peuvent pas être interprétées directement par les transformations de modèles dans un modèle MARTE, en raison de l'absence de méta-classes correspondantes dans le méta-modèle actuel MARTE.

Ainsi, ces concepts ont été introduits dans une version étendue du méta-modèle MARTE, en vue de permettre une bonne interprétation. Tout d'abord, un élément UML, appartenant au profil MARTE, modélisé grâce à un outil de modélisation graphique tel que Papyrus, se transforme en un StructuredComponent MARTE. Cette entité contient des éléments supplémentaires, tels que les p*arts*, les *flow ports* et *connecteurs*.

Étant donné que ces concepts ont été développés par un effort collectif au sein de l'équipe DaRT et ne sont donc pas notre contribution particulière de mes travaux, elles sont représentées différemment dans la figure 4.2, par rapport à mes propres contributions (présentées d'une manière normale).

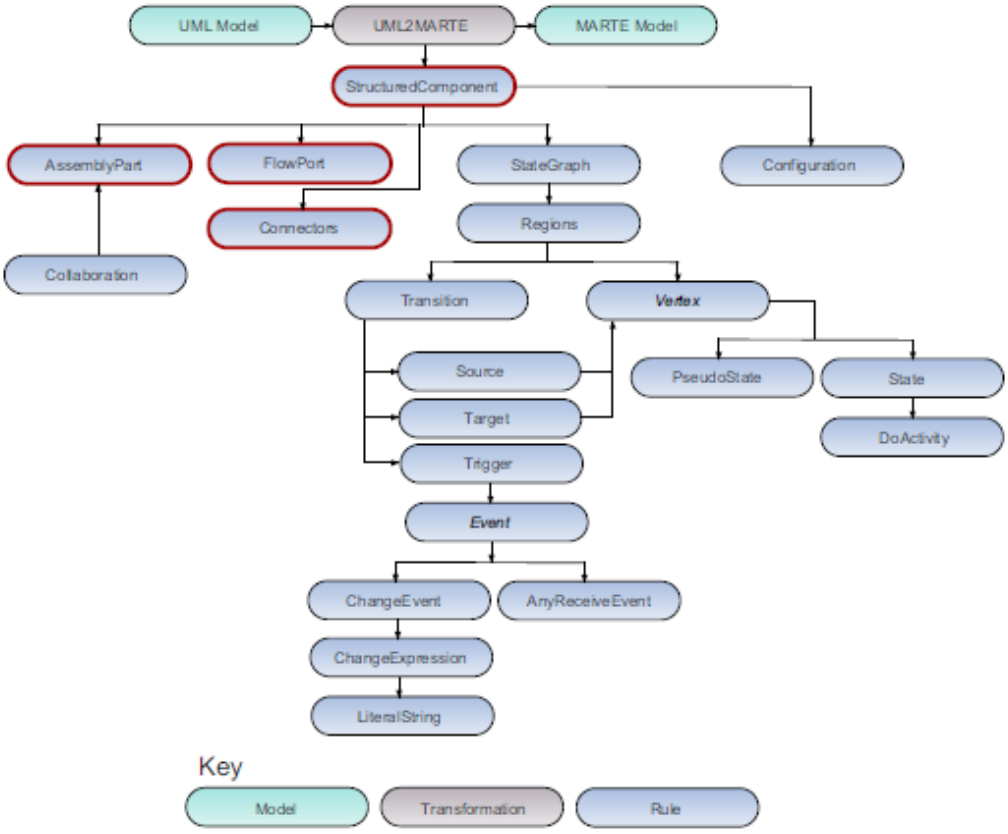


Figure 4.2: Aperçu de la transformation *UML2MARTE*

La figure 4.2 montre un aperçu global de cette transformation de modèles, à l'égard du flot de conception Gaspard2. Dans *UML2MARTE*, nous transformons d'abord une machine d'états UML (liée à un composant) en un graphe d'états lié à au composant structuré correspondant. Un graphe d'états peut avoir plusieurs régions, chacune pouvant avoir de multiples transitions et des sommets. Un sommet est un concept abstrait et peut être soit un pseudo état ou un état. Un état peut avoir un *doActivity* qui détermine son comportement. Une transition est constituée des sommets source et cible, et peut avoir plusieurs déclencheurs. Chaque déclencheur est associé à un événement, soit un *ChangeEvent* ou un *AnyReceiveEvent*. Dans le premier cas, il contient un *ChangeExpression* qui est effectivement transformé en un *LiteralString* MARTE tel que spécifié dans le package de VSL. Les collaborations UML, présentes dans le modèle UML sont directement converties en collaborations MARTE dans le modèle MARTE. Une collaboration dans notre chaîne de transformation peut avoir des *parts* d'assemblage intérieures, qui servent à déterminer le comportement du composant structuré, auquel la collaboration se rapporte.

Enfin, une configuration présente dans le modèle UML sous la forme d'un composant est également convertie en un élément structuré et contient des informations relatives à l'implémentation logicielle dans le modèle UML.

4.3.2. Transformations MARTE2RTL

Une fois le modèle MARTE créé, à partir de la transformation *UML2MARTE*, le modèle obtenu est transformé dans un modèle RTL en utilisant *MARTE2RTL*.

La transformation *MARTE2RTL* prend le modèle MARTE en entrée et génère deux modèles de sortie: le modèle RTL, contenant les concepts liés à l'accélérateur matériel et le contrôle, et le modèle RTL *PortType*, contenant le contrôle et les types de données présentes dans le modèle MARTE. La chaîne de transformation *MARTE2RTL* a été entièrement développée au cours de mes travaux dans la cadre de la thèse [QAD08]. Les règles de transformations liées au modèle RTL *PortType* permettent la création de différents types de contrôle et de données relatifs aux ports des concepts de contrôle et à la fonctionnalité de l'application. Ainsi, trois types sont créés: *RTL Primitive Type*, *RTL Complex Type* et *RTL Enumeration*.

En ce qui concerne le modèle de RTL, les règles significatives sont: AMARTE *StructuredComponent* ayant des instances de composants multiples et aucune collaboration associée n'est transformée en un élément *HW Compound*, tandis qu'un composant structuré similaire ayant des collaborations liées (qu'elles soient liées à lui-même ou à un composant d'un sous niveau hiérarchique) est converti en un *Compound component* de contrôle. Les collaborations qui sont présentes dans le modèle MARTE sont directement converties en *counter parts* équivalentes.

Toutes les transformations sont détaillées de façon exhaustive dans [QAD08]. Cependant, je souhaiterai souligner ici que la quasi équivalence entre les règles relatives aux graphes d'états dans les transformations *UML2MARTE* et leurs correspondances dans *MARTE2RTL* permettent des transformations un pour un assez systématiques.

4.3.3. Les choix possibles pour la génération de code

Un modèle RTL n'est pas directement exploitable par les outils classiques de la simulation et de synthèse, contrairement à un code exécutable produit. Ainsi, il y a deux approches possibles pour la génération de code, qui sont expliquées ci-dessous:

Première solution: la création de deux méta-modèles distincts respectant les syntaxes des langages VHDL et C/C++ respectivement. Ceci implique deux autres transformations

modèle à modèle, à partir du modèle RTL vers les modèles VHDL et C/C++. Ainsi, à partir de ces derniers, la génération de code peut être réalisée.

Deuxième solution: la génération de code directement à partir du modèle RTL. L'avantage de cette méthode est qu'aucun méta-modèle intermédiaire n'est nécessaire dans notre flot de conception. En effet, le niveau d'abstraction du modèle RTL est suffisamment bas pour permettre la génération de code VHDL et C/C++.

La première solution présente un certain nombre d'inconvénients. Tout d'abord, l'effort de développement est augmenté avec l'introduction de méta-modèles supplémentaires et les transformations correspondantes, dans le flot de conception. En outre, si l'on veut générer le code d'un accélérateur matériel, dans une autre syntaxe (comme Verilog), il faudra alors créer encore un méta-modèle supplémentaire concernant le langage cible. De même, pour la partie contrôleur de reconfiguration, comme expliqué précédemment, le code correspondant à sa machine d'état peut être généré en C/C++ ou VHDL selon le type d'implémentation souhaité (logicielle ou matérielle). Si les deux méta-modèles C/C++ et VHDL sont implémentés, alors soit les concepts de contrôleur doivent être intégrés dans les deux méta-modèles (redondance), ou bien uniquement dans l'un des deux méta-modèles. Cela diminuerait la flexibilité de notre flot de conception.

Pour ces raisons, la deuxième solution a donc été adaptée. Ceci entraîne la génération de code directement à partir du modèle RTL, au moyen de la transformation *RTL2CODE*. Cette solution est assez générique, et diminue également les efforts de développement et de maintenance des développeurs.

4.3.4. Génération de code VHDL pour les accélérateurs matériels

L'un des objectifs du flot de conception est la génération de code VHDL synthétisable et correct pouvant être pris en entrée par des outils commerciaux de synthèse, pour l'implémentation sur un FPGA. Ce code VHDL correspond aux implémentations de l'application modélisée, ou les différentes configurations d'un accélérateur matériel dynamiquement reconfigurable.

Pour que le code généré ait les mêmes caractéristiques, telles que le parallélisme de données et de tâches, qui étaient présentes dans le modèle d'application initial, nous avons imposé certaines conditions qui sont prises en compte lors de l'écriture des templates JET. Certaines des conditions critiques pour la génération de code sont données ci-dessous:

Structure de l'accélérateur: La structure de l'accélérateur matériel décrit en VHDL doit être équivalente à l'application modélisée au niveau du profil MARTE. Ainsi la hiérarchie de l'application est conservée dans le code final.

Génération de code pour les composants de l'accélérateur: Chaque élément dans une configuration de l'accélérateur matériel est écrit dans un fichier distinct, dont le nom correspondant à celui du composant.

Parallélisme exprimé: Le parallélisme de données des applications Gaspard2 est exprimé dans l'accélérateur en utilisant le mot clé VHDL GENERATE. De même, les ports multidimensionnels ne sont pas linéarisés au cours de la phase de génération de code.

Configurations: Selon le nombre de configurations au niveau de la modélisation de haut niveau, nous trouvons le même nombre d'implémentations de l'accélérateur matériel. Un dossier distinct est créé pour chaque configuration/implémentation de l'accélérateur matériel. Bien que, pour chaque configuration, une grande partie du code VHDL généré reste le même, le code correspondant aux composants élémentaires est distinct pour chaque configuration.

Ce choix a été fait pour lever l'ambiguïté et faciliter la création de bitstreams partiels. Bien qu'il soit possible de créer un seul accélérateur matériel et de modifier manuellement les implémentations liées aux composants élémentaires, et ensuite créer des configurations différentes d'une manière non automatique, ceci s'avère être est une tâche fastidieuse qui augmente la complexité en fonction de l'augmentation du nombre de composants élémentaires ou de configurations.

4.3.4.1. Génération du code du contrôleur de reconfiguration

Le code généré pour le contrôleur de reconfiguration est en C/ C++, en raison du choix d'utiliser pour son exécution, un processeur interne intégré dans le FPGA cible. Ainsi, l'automate modélisé est transformé en une machine d'état en C/C++, avec des transitions continues infinies, pour être une description équivalente aux automates de mode.

4.3.5. Thèses et masters

Le travail présenté dans cette section à été réalisé dans le cadre de la thèse [QUA10].

4.3.6. Publication représentative

Titre
Targeting Reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation
Publication
Int. J. Embedded Systems (IJES)
Auteurs
Imran Rafiq Quadri, Abdoulaye Gamatié, Samy Meftali, Jean-Luc Dekeyser, Huafeng Yu, Eric Rutten
Résumé
Comme la complexité de la conception de systèmes sur puce est sans cesse en hausse, il ya un besoin urgent aujourd'hui de trouver des approches et des outils adéquats la manipulation et la production des SoCs. En outre, les SoCs modernes reconfigurables offrent des avantages considérables par rapport SoCs classiques comme ils intègrent les caractéristiques d'adaptabilité pour faire face aux changements éventuels des applications et del' environnement.
Cet article présente une nouvelle approche pour traiter adaptabilité et la reconfiguration des systèmes. Un modèle générique de contrôle réactif est présenté dans le cadre de l'environnement de co-design Gaspard2. Une comparaison de l'intégration du contrôle à différents niveaux du flot est réalisée.
Les travaux présentés sont basés sur l'approche IDM et le profil UML MARTE proposé par l'Object Management Group, pour la modélisation et l'analyse des systèmes temps réel embarqués. Nos contributions consistent à présenter un flot de conception complet pour passer de modèles de haut niveau MARTE jusqu'à la génération automatique de code, pour l'implémentation des systèmes sur puce reconfigurables dynamiquement.

4.3.7. Analyse et discussion

Cette section présente mes contributions liées aux transformations modèle à modèle implémentées dans notre flot de conception. Ainsi, les concepts introduits dans le chapitre 2 sont transformés par les deux transformations modèles à modèle: UML2MARTE et MARTE2RTL vers un modèle RTL.

Une fois les transformations modèle à modèle ont été exécutées, le modèle RTL résultant fournit une estimation précise des détails relatifs au niveau transfert de registre. De même, les concepts de contrôle sont suffisamment enrichis pour permettre la génération de code. La transformation RTL2CODE, permet de générer le code pour les différentes implémentations d'un accélérateur matériel reconfigurable dynamiquement, ainsi que le code pour la gestion de reconfiguration.

Ces contributions ont ainsi ouvert la possibilité de cibler les architectures FPGAs modernes, pouvant être dynamiquement et partiellement reconfigurées. En effet, l'état actuel

de l'outil Gaspard2 permet uniquement la génération de code pour des contrôleurs de reconfiguration dépourvus d'intelligence. Le code généré avec RTL2CODE vise des implémentations sur des FPGAs de Xilinx et il est donc utilisable par les outils de synthèse propriétaires correspondants.

Les travaux sur les outils d'aide à la conception ciblant les codes d'implémentation continueront certainement à faire partie des priorités de mes travaux de recherche futurs. Car il faudra maintenir, sans cesse, les outils développés et les adapter aux futurs standards en termes de langages et outils de transformation notamment. Ainsi, mes contributions futures dans Gaspard2 viseront à court terme de produire du code pour les nouveaux outils annoncés chez Altera et Xilinx.

4.4. Méta-modèle SystemC pour la génération de code de simulation

4.4.1. Introduction

La toute première version de l'environnement Garpard permettait uniquement de modéliser des applications et des architectures pour réaliser une association.

Les techniques de co-simulation étaient donc exclues. Ceci empêchait toute validation des systèmes modélisés dans Gaspard. Ainsi, ma contribution dans cette version de l'outil était la définition et l'implémentation de la partie basse du modèle Y.

A cette époque, il n'existait aucun profil ou méta-modèle standard permettant de cibler un langage de co-simulation dans une démarche IDM. Le but était donc de définir un méta-modèle regroupant les concepts logiciels, matériels et des mécanismes de synchronisation pour produire, par transformations automatiques modèle à modèle, du de code SystemC au niveau TLM.

4.4.2. Concepts du méta-modèle

Le but est de réaliser des simulations de haut niveau en SystemC en générant, automatiquement, le code dans l'environnement Gaspard. Pour cela, on à défini un méta-modèle UML, basé sur SystemC, permettant d'exécuter des applications de traitements de signal intensif. Ces applications étant elles-mêmes définies par ISP-UML, un autre méta-modèle défini par l'équipe DaRT.

Le méta-modèle contient un ensemble de concepts que l'on peut classer en deux catégories : concepts logiciels et concepts matériels.

Ceux de la première catégorie servent à modéliser tous les aspects liés à l'application. Les *Tilers* et les *Tâches* sont les deux concepts clés de cette classe.

Dans la seconde catégorie, on trouve principalement les concepts de : *Processeur, Mémoire, FIFO, Bus, NoC* et *Bridge.*

A partir de ces concepts, nous sommes capables de simuler des applications de traitement du signal sur des architectures matérielles, en utilisant en plus, la notion de tableaux.

Ainsi, les mémoires contiennent les éléments des différents tableaux que l'on manipule. Les tilers permettent, soit de transformer des données provenant d'une mémoire en un nouveau tableau appelé motif, soit de découper un motif pour mettre les données dans une nouvelle mémoire. Cela implique que les tilers doivent réaliser des calculs d'adresses à partir de données provenant de l'application définie en ISP-UML. Les FIFOs permettent de stocker des motifs avant traitement par la tâche ou alors après traitement par celle-ci. Enfin les tâches sont nos éléments de traitement des données, ce sont elles qui réalise les différentes opérations sur les motifs. Ils existent deux types de tâches, les tâches dites élémentaire, qui ne font qu'exécuter un bout de code que nous leurs avons fournis et des tâches composées d'une ou plusieurs tâches (élémentaires ou composées) reliées par des FIFOS, tilers et des mémoires. L'ensemble des tâches composant le système est mit à plat dans le modèle de simulation est exécuté sur des processeurs.

En plus des concepts décris ci-dessus, le méta-modèle SystemC contient également des éléments dédiés à la synchronisation, tels que les ports et signaux de contrôle.

4.4.3. Critères pour l'estimation des performances

On peut imaginer un très grand nombre de critères de performance pour un système, dépendamment de son utilisation et de l'environnement où il évolue.

Pour obtenir un méta-modèle permettant une génération de code et dans un souci simplicité (comme il s'agissait de la première version d'un tel méta-modèle), on a réduit le nombre de critères possibles à quatre [SMN04] :

– puissance (Power)

– taille (Space)

– temps d'exécution (Time)

– coût (Cost)

Avec *puissance*, j'entends bien sûr la consommation d'énergie du système. Ce critère est critique dès lors que l'on parle de système embarqué. En effet, la plupart de ces systèmes fonctionnent de manière autonome, sur batterie par exemple, ce qui rend primordial la réduction de la consommation d'énergie. J'ai d'ailleurs poursuivi mes travaux sur ce point comme décrit dans le chapitre 3.

L'évaluation du temps d'exécution du système final est également un point très important dans la conception de systèmes sur puce. En effet, il s'agit d'un paramètre crucial pour la réalisation d'applications telles que des codecs vidéo. Pour cela, il me paraissait nécessaire d'ajouter un critère permettant cette évaluation.

Un autre critère important, dans le monde des systèmes embarqués, est la taille. Un des objectifs principaux des concepteurs de ces systèmes est de fournir l'architecture la plus performante dans l'espace le plus réduit possible.

Pour terminer avec la présentation de ces critères, le critère de coût permet d'évaluer le coût, en terme financier, du système (achat d'IP, etc.).

Il est vrai que l'estimation des deux derniers critères peut être faite durant la phase de modélisation, mais il me semble judicieux de les faire tout de même apparaitre car ils peuvent être d'une grande utilité si l'on veut faire, par exemple, de l'exploration d'architectures.

4.4.4. Thèses et masters

Ce travail a été entièrement réalisé dans le cadre de la thèse de Mickael Samyn [SAM05] que j'ai codirigé à 50%. Elle fut l'un des premiers travaux la définition et l'intégration d'un méta-modèle SystemC dans un environnement IDM.

4.4.5. Publication représentative

Titre
MDA Based, SystemC Code Generation, Applied to Intensive Signal Processing Applications
Publication
FDL'04. Septembre 2004
Auteurs
Mickael Samyn, **Samy Meftali**, Jean-Luc Dekeyser
Résumé
Dans cet article, nous présentons une nouvelle méthodologie pour la conception de systèmes sur puce dédiés aux applications de traitement de signal intensif. Nous générons de manière systématique et automatique, une spécification de SystemC au niveau transactionnel en utilisant l'approche MDA.
Le code généré décrit l'application mappée sur une architecture multiprocesseur. En plus des processeurs, cette dernière peut intégrer plusieurs mémoires et un réseau d'interconnexion générique et configurable.
L'efficacité de notre méthodologie a été montrée sur un exemple d'application de traitement de signal intensif.

4.4.5. Analyse et discussion

Ces travaux sur la génération automatique de code de simulation (SystemC), en utilisant une approche IDM, étaient précurseurs au sein de l'équipe DaRT en termes d'intégration de cibles de simulation de matériel dans Gaspard. Ils furent même parmi les tout premiers travaux ciblant la génération de code SystemC dans un environnement IDM pour la conception conjointe logicielle/matérielle.

Depuis, certains travaux dans l'équipe ont été orienté dans le même sens. Il s'agit principalement d'efforts pour générer du code conforme au standard TLM. Les nouveaux travaux de l'équipe ont introduits le concept de déploiement d'IPs. En effet, la génération de code dans Gaspard2 permet d'instancier des composants de simulation provenant d'une bibliothèque. Ceci rend la production de code plus systématique et moins fastidieuse. En plus, certain de mes travaux décrits dans le chapitre 3 intègrent une estimation efficace de la consommation d'énergie dans les modèles de simulation.

Cependant, je pense qu'il reste encore beaucoup de travail sur cet aspect central dans tout outil de conception moderne, qui est la génération de code de simulation. Ainsi, les travaux futurs iront dans le sens de la conception à base de FPGAs dynamiquement reconfigurables. En effet, je suis convaincu qu'il serait très difficile de se passer de simulations SystemC (ou langages équivalents) supportant la reconfiguration partielle et dynamique dans les futurs outils de conception.

L'intérêt de la simulation n'est plus à discuter pour ce qui concerne notamment l'exploration d'architectures. Mais une fois le concepteur satisfait du système simulé, vient forcement l'inévitable étape du passage à une spécification au niveau VHDL (ou Verilog) synthétisable, en vue de l'implémentation du système. C'est à ce moment là qu'un problème difficile mais important fait son apparition. Il s'agit de s'assurer de la conformité du code produit pour la synthèse avec le code simulé.

Je pense donc que les méthodologies de vérification et de démonstration d'équivalence entre spécifications constitueront un point clé dans la conception lors des prochaines années.

Chapitre 5: Conclusion et perspectives

5.1. Bilan

Ce manuscrit fait un bilan de huit ans de recherche. Si mes travaux s'inscrivent toujours dans le domaine de la conception conjointe logicielle/matérielle, trois grands axes ont été explorés : la modélisation (et méta-modélisation) des systèmes, les environnements de simulation multi-niveaux distribués avec estimation de la consommation d'énergie et la production d'outils de conception basés sur l'approche IDM.

La complexité grandissante des SoCs, oriente de façon inéluctable les concepteurs d'aujourd'hui vers des approches de conception partant de modèles de très niveau. Car il serait tout à fait inenvisageable de manipuler un système, avec des centaines de millions de transistors, directement à partir de bas niveaux d'abstraction comme le RTL.

L'apparition, lors des dernières années de standards reconnus comme UML2 [OMG03], le profil MARTE et l'approche IDM a accéléré considérablement les travaux autours de la modélisation de haut niveau. En effet, ces standards fournissent un atout important, qui est la possibilité d'échange et réutilisation systématiques des modèles développés. Mes travaux sur ce thème ont commencé depuis cinq ans. Ils ont aboutis à la proposition de méta-modèles et profils pour le niveau RTL ciblant les FPGAs reconfigurables, méta-modèle de contrôle au niveau déploiement pour la modélisation de contrôleurs de reconfiguration, ainsi que l'intégration de l'estimation de la consommation des FPGAs dans le profil MARTE.

J'ai accordé aux environnements de simulation des systèmes logiciel/matériel un grand intérêt dans mes travaux de recherche depuis 2002. En effet, il me semble qu'ils constituent un défit majeur dans tout outil de conception, vu l'importance de la simulation et la part du temps de conception qui lui est consacré.

Ainsi, trois aspects me paraissent indispensables dans un environnement de simulation efficace. Il doit supporter des IPs décrits à différents niveaux d'abstraction, faire intervenir plusieurs machines dans une même simulation et pouvoir produire de bonnes estimations de la consommation d'énergie pour une exploration d'architectures efficace.

Ces trois points ont ainsi constitué mes thématiques principales de recherche sur les environnements de simulation. Ainsi, j'ai présenté dans ce mémoire des solutions pour la communication entre IPs de simulation décrits à différents niveaux d'abstraction pour des simulations multi-niveaux ; une solution basée sur CORBA pour la simulation distribuée, où l'affectation des IPs sur les machines est optimisée en utilisant une méthode combinatoire exacte ; et enfin deux démarches pour l'intégration de l'estimation de la consommation d'énergie dans les modèles de simulation au niveau CABA. La première étant basée sur l'intégration d'un code spécifique aux modèles et la seconde, est non intrusive, se basant sur le principe d'observation par espionnage.

Le troisième axe de mes travaux de recherche consiste à proposer des méthodes et outils automatiques pour la conception logicielle/matérielle. Mes contributions sur ce point se répartissent sur deux périodes.

La première période de 2002 à 2005 concernait la toute première version de l'environnement Gaspard. En effet, la version de Gaspard à l'époque ne permettait que de modéliser une application, une architecture ainsi que leur association. Mes travaux de recherche ont donc permit de compléter l'outil en rajoutant une phase de génération de code SystemC pour la simulation fonctionnelle. Le code généré intégrait tous les aspects liés à la synchronisation des tâches et les calcules d'adresses pour les accès mémoires.

La seconde période allant de 2006 à aujourd'hui concerne la version 2 de l'environnement Gaspard. Mes travaux se sont alors concentrés sur des méta-modèles et leurs transformations automatiques pour la génération de code synthétisable au niveau RTL. Ainsi, je me suis intéressé à la génération de contrôleurs de reconfiguration et de spécifications ciblant des implémentations sur FPGAs dynamiquement reconfigurables.

5.2. Discussion

Les travaux sur les méthodes et outils de conception logicielle/matérielle ont fait leur apparition depuis le début des années 90. Cependant, aucun outil académique ou commercial, aujourd'hui, ne semble se dégager comme solution complète et satisfaisante. De ce fait, on peut s'interroger légitimement sur les raisons de ce constat.

En fait un projet de conception fait intervenir plusieurs types de métiers, pouvant être de natures très différentes. En effet, le concepteur d'application, l'architecte matériel, les ingénieurs de validation et de tests ainsi que les intégrateurs doivent y collaborer. La cohabitation entre tous ces métiers n'est sans doute pas une tâche aisée.

Ainsi, l'expérience nous apprend aujourd'hui, que réaliser un tel outil en partant de zéro (from scratch) ne serait pas une solution réaliste ni réalisable. En effet, l'idéal à mon sens serait de réfléchir plutôt sur une intégration automatique et systématique d'outils existants (autant que possible). Ceci permettra à la fin de ne se concentrer que sur des parties spécifiques et dédiées à un domaine particulier (comme un domaine d'application spécifique ou des architectures matérielles particulières comme les FPGAs dynamiquement reconfigurables.

L'utilisation des standards comme OCP pour les interfaces de communication et le profil MARTE pour la modélisation ma semble être un moyen intéressant pour aboutir un outil de conception efficace. En effet, ces standards augmentent considérablement les possibilités de réutilisation, et rendent l'interfaçage entre outils divers assez systématique et beaucoup moins fastidieux.

Une des solutions est de posséder un environnement unique, construit autour de l'approche IDM [OMG07], permettant de modéliser des applications et architectures (intégrant éventuellement des FPGAs dynamiquement reconfigurables). Cet outil doit être capable, en partant des mêmes modèles de haut niveau, de manipuler plusieurs niveaux d'abstraction et cibles d'exécutions, pour viser la simulation, le prototypage ou la réalisation d'un circuit. Gaspard répond à un certain nombre de ces objectifs, et tous les travaux de l'équipe DaRT visent à le compléter dans ce sens.

5.3. Perspectives et évolution

Le succès grandissant des technologies FPGAs, et l'émergence de plateformes performantes supportant la reconfiguration partielle et dynamique, ne fait que conforter mon choix d'orienter mes travaux sur la conception de tels systèmes.

5.3.1. Conception d'un environnement de conception complet, ciblant les FPGAs partiellement et dynamiquement reconfigurables

Il s'agit là d'un sujet prioritaire pour mes travaux de recherche dans les prochaines années. En effet, j'ai monté le projet ANR FAMOUS dans ce sens là. Il s'agit d'un projet de 48 mois, regroupant des partenaires académiques et industriels, débuté en décembre 2009. Il vise à présenter une méthodologie complète qui prend en compte la reconfiguration

dynamique du matériel, et propose les mécanismes nécessaires pour exploiter entièrement ces possibilités pendant l'exécution.

FAMOUS s'intéresse aux modèles de très haut niveau (UML), méthodes de compilation et d'exécution ainsi que les techniques d'analyse et de vérification. L'objectif est de fournir des outils pour une conception de qualité, améliorants la productivité, tout en garantissant l'optimisation des ressources matérielles utilisées et en réduisant le temps de mise sur le marché.

Les contributions principales de FAMOUS se situent à plusieurs niveaux :

- Modélisation de haut niveau : le projet vise à fournir un environnement de modélisation dédié aux applications de traitement d'image ciblant des architectures dynamiquement reconfigurables. La partie modélisation sera basée sur le standard MARTE, mais en lui intégrant tous les concepts manquant pour la modélisation de ce type de systèmes. Ainsi, cet environnement se doit de permettre de modéliser la reconfiguration dynamique à la fois au niveau application, architecture et déploiement. En effet un système à base d'architectures reconfigurables implique d'éventuelles modifications dans le modèle de l'application (changement de tâches), modification de l'architecture en termes de topologie par exemple, et du déploiement sur les IPs d'implémentation.
- Modèle d'exécution dynamique pour FPGAs dynamiquement reconfigurables : la modélisation d'architectures dynamiquement reconfigurables, même de grande qualité ne peut, à mon avis, être efficace si l'ont ne dispose pas d'un modèle d'exécution clairement défini et dédié à ce type d'architectures. Un tel modèle devrait prendre en charge tous les aspects liés à la granularité de la reconfiguration, du changement de contexte, etc.
- Bibliothèque d'IPs de simulation et d'implémentation dédiée à la reconfiguration dynamique : il s'agit là d'une bibliothèque d'IPs logiciels et matériels dédiés aux applications de traitement d'image et aux architectures à base de FPGAs partiellement et dynamiquement reconfigurables. Ces IPs devraient intégrer des informations liées à l'estimation des performances et de la consommation d'énergie.
- Transformations automatiques modèle à modèle : on vise dans le projet FAMOUS à utiliser des outils de transformations automatiques génériques pour faire le pont entre les divers modèles et méta-modèles du flot.

5.3.2. Approche par espionnage pour le contrôle de la reconfiguration dynamique

Les caractéristiques de non intrusivité de la méthode d'estimation de la consommation présentée au chapitre 3, ainsi que sa généricité (utilisation d'interfaces standards pour permettre l'automatisation), me rendent convaincu la généralisation d'une telle approche et son adaptation au contexte de la reconfiguration dynamique pourraient donner des résultats intéressants.

Ainsi, le projet que j'ai lancé dans le cadre de la thèse de Chiraz Trabelsi en octobre 2009, consiste à définir une méthodologie permettant de modéliser et de générer automatiquement des contrôleurs de reconfiguration dynamique distribués. Le but est que ces contrôleurs soient simples et légers. Chacun observant une partie du FPGAs pour minimiser leur part dans la complexité du système et de sa consommation d'énergie. Ainsi, la décision de reconfigurer le système se prend par la collaboration de tous ces contrôleurs répartis.

5.3.3. Environnement de simulation et d'estimation de performances, pour architectures dynamiquement reconfigurables

Il s'agit là, à mon sens, de l'un des défit majeurs pour l'environnement de conception futurs. En effet, la réduction du temps de simulation, l'interopérabilité et la précision ainsi que la pertinence des observations pour une éventuelle utilisation lors de l'exploration d'architecture me semblent constituer des verrous scientifiques intéressants.

Ainsi, j'envisage de continuer à l'intéresser dans mes travaux futurs à des environnements de simulation basés entièrement sur l'approche IDM. Ces derniers devraient fournir les IPs et les mécanismes nécessaires pour la simulation d'architectures partiellement et dynamiquement reconfigurable. Car je suis convaincu aujourd'hui que ce type d'architectures constituera dans le futur la cible, par excellence, des prototypages.

Un tel environnement devrait intégrer une méthodologie d'estimation de la consommation dédiée aux FPGAs dynamiquement reconfigurables. Ceci permettra par la suite une utilisation efficace pour l'exploration rapide des architectures, avant les phases d'implémentation.

Références bibliographiques

[AMA07a] Y. Aydi, S. Meftali, M. Abid, J.-L. Dekeyser. Dynamicity Analysis of Delta MINs for MPSoC Architectures, in: Conférence internationale des Sciences et Techniques de l'Automatique (STA'07), November 2007

[AMA07b] Yassine Aydi, Samy Meftali, Mohamed Abid, Jean-Luc Dekeyser. Design and Performance Evaluation of a Reconfigurable Delta MIN for MPSOC. In 19th International Conference on Microelectronics (ICM'07), Cairo, Egypt, December 2007.

[AMB] AMBA System Architecture, http://www.arm.com/products/solutions/AMBAHomePage.html

[AnNa04] Jason H. Anderson, Farid N. Najm, Power estimation techniques for FPGAs , IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Issue 10 (October 2004)

[BBN06] R. B. ATITALLAH, L. BONDE, S. NIAR, S. MEFTALI, J.-L. DEKEYSER. Multilevel MPSoC performance evaluation using MDE approach., in: International Symposium on System-on-Chip 2006, Tampere, Finland, november 2006

[BDL07] P. Marquet, S. Duquennoy, S. Le Beux, S. Meftali, J.-L. Dekeyser. Massively Parallel Processing on a Chip, in: ACM Int'l Conf. on Computing Frontiers, Ischia, Italy, May 2007

[BEN08] R. Ben Atitallah, Modèles et simulation des systèmes sur puce multiprocesseurs- Estimation des performances et de la consommation d'énergie, Thèse. Université des sciences et technologies de Lille, Mars 2008.

[BER08] Berthelot et al. A Flexible system level design methodology targeting run-time reconfigurable FPGAs. EURASIP Journal of Embedded Systems, 2008.

[BeRo97] Vaughn Betz et Jonathan Rose. VPR: A new packing, placement and routing tool for fpga research. In FPL 1997: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, pages 213–222, London, UK, 1997.

[BeRo99] V. Betz, J. Rose et A. Marquardt. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Press, 1999.

[BEZ04] Jean Bézivin. In search of a basic principle for model driven engineering. The European Journal for the Informatics Professional, April 2004.

[BMM05] J.-L. Dekeyser, P. Marquet, S. Meftali, C. Dumoulin, P. Boulet, S. Niar. Why to do without Model Driven Architecture in embedded system codesign?, in: first annual IEEE BENELUX/DSP Valley Signal Processing Symposium, SPS-DARTS 2005, Antwerp, Belgium, April 2005.

[BMP07] P. Boulet, Ph. Marquet, E. Piel, J. Taillard. Repetitive Allocation Modelling with MARTE. In Proceedings of the FORUM on Specification & Design Languages, Barcelona, Spain, September 18-20, 2007.

[BND07] Rabie Ben Atitallah, Smail Niar, and Jean-Luc Dekeyser. MPSoC Power Estimation Framework at Transaction Level Modeling. In The 19th International Conference on Microelectronics (ICM 2007), Cairo, Egypt, December 2007.

[BNG06] R. B. Atitallah, S. Niar, A. Greiner, S. Meftali, J. L. Dekeyser. Estimating Energy Consumption for an MPSoC Architectural Exploration, in: ARCS'06: Architecture of Computing Systems, Frankfurt, Germany, March 2006.

[BNM06] R. B. ATITALLAH, S. NIAR, S. MEFTALI, J.-L. DEKEYSER. Accelerating MPSoC Performance Evaluation with TLM., in: "Advanced Computer Architecture and compilation for Embedded Systems ACACES", HIPEAC network of Excellence summer school, Laquilia, Italy, ISBN 90 382 0981 9, july 2006, p. 229–232

[BNM07] R. Ben Atitallah, S. Niar, S. Meftali, J.-L. Dekeyser. An MPSoC performance estimation framework using transaction level modeling, in: IEEE RTCSA'2007, Daegu, Korea, 2007

[BNR06] Alistair Bruce, Andrew Nightingale, Nizar Romdhane. Maintaining Consistency Between SystemC and RTL System Designs. Proceedings of the 43rd annual Design Automation Conference. San Francisco, CA, USA 2006.

[BON06] L. Bondé, Transformations de Modèles et Interopérabilité dans la Conception de Systèmes Hétérogènes sur Puce à Base d'IP. Thèse. Université des Sciences et Technologies de Lille, Décembre 2006.

[BOU07] P. Boulet. Array-OL revisited, multidimensional intensive signal processing specification. Technical Report RR-6113, Feb. 2007.

[BPL01] Bakshi, V.K. Prasanna, A. Ledeczi. MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems. In Workshop on Languages, Compilers, and Tools for Embedded Systems, LCTES 2001.Snowbird, Utah, June 2001.

[BPN07] Rabie Ben Atitallah, Eric Piel, Smail Niar, Philippe Marquet, and Jean-Luc Dekeyser. Multilevel MPSoC simulation using an MDE approach. In IEEE International SoC Conference (SoCC 2007), Hsinchu, Taiwan, September 2007.

[BPT07] Rabie Ben Atitallah, Eric Piel, Julien Taillard, Smail Niar, and Jean-Luc Dekeyser. From High Level MPSoC description to SystemC Code Generation. In International Mod-Easy'07 Workshop in conjunction with Forum on specification and Design Languages (FDL'07), Barcelona, Spain, September 2007.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi. Wattch : a framework for architectural-level power analysis and optimizations. In Proceedings of the 27th annual international symposium on Computer architecture, pages 83–94, 2000.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi. Wattch : a framework for architectural-level power analysis and optimizations. In Proceedings of the 27th annual international symposium on Computer architecture, pages 83–94, 2000.

[CaGa03] L. Cai and D. Gajski. Transaction level modeling: an overview. In CODES+ISSS'03, New York, USA. 2003.

[CCZ06] Chopard, B., Combes, P., Zory, J.: A parallel version of the osci systemc kernel. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 653–660. Springer, Heidelberg (2006)

[CDG07] Sumanta Chaudhuri, Jean-Luc Danger et Sylvain Guilley. Efficient Modeling and Floorplanning of Embedded -FPGAFabric. In 17th IEEE International Conference on Field Programmable Logic and Applications, FPL, Amsterdam, Netherlands, Août 2007.

[CHE09] Sana Cherif "Extension du méta-modèle d'architecture de MARTE : Modélisation des FPGAs et prise en compte de leur consommation d'énergie", Mémoire de master de l'INSAT Tunis, novembre 209.

[ChOr95] P. H. CHOU, R. B. ORTEGA, et AL. «The Chinook Hardware/Software Co-Synthesis System ». Dans *Proceedings of the International Symposium on System Synthesis*, 1995.

[CJM02] S. Choi, J.W. Jang, S. Mohanty, V. K. Prasanna. Domain-Specific Modeling for Rapid System-Level Energy Estimation of Reconfigurable Architectures. In Proceedings of International Conference of Engineering of Reconfigurable Systems and Algorithms, ERSA, June 2002, Las Vegas, Nevada, USA.

[COR] CoreConnect bus architecture, http://www3.ibm.com/technology/power/licensing/coreconnect/index.html

[COW02] Coware. Inc., "N2C" available at http://www.coware.com/cowareN2C.html

[COW10] CoWare® Platform Architect. http://www.coware.com/

[COX05] Cox, D.: Ritsim: Distributed systemc simulation. Master's thesis, Rochester Institute of Technology (2005), http://hdl.handle.net/1850/1014

[Dar05] DaRT Team LIFL, Lille, France. Graphical Array Specification for Parallel and Distributed Computing (GASPARD-2). http://www2.lifl.fr/west/gaspard/, 2005.

[DGM10] Jean-Luc Dekeyser, Abdoulaye Gamatié, Samy Meftali and Imran Rafiq Quadri (In alphabetical order) "High-level modeling of dynamically reconfigurable heterogeneous systems", Chapter 7 in Heterogeneous Embedded Systems - Design Theory and Practice: Springer, Tentative publication date: 2010.

[DHO98] Dongarra, J., Huss-Lederman, S., Otto, S., Snir, M., Walkel, D.: The message passing interface (mpi) standard (1998), http://www-unix.mcs.anl.gov/mpi

[DLM06] S. DUQUENNOY, S. LE BEUX, P. MARQUET, S. MEFTALI, J. DEKEYSER. MpNoC Design: Modeling and Simulation, in: 15th edition of IP/SOC, 2006

[DMM05] Jean-Luc Dekeyser, Philippe Marquet, Samy Meftali, Cédric Dumoulin, Pierre Boulet, and Smail Niar. Why to do without Model Driven Architecture in embedded system codesign?. In The first annual IEEE BENELUX/DSP Valley Signal Processing Symposium, SPS-DARTS 2005, Antwerp, Belgium, April 2005.

[EDR07] A. Etien, C. Dumoulin et E. Renaux. Towards a unified notation to represent model transformation. Rapport de recherche RR-6187, INRIA, mai 2007.

[ELL06] D. Elléouet. Méthodes de modélisation, d'estimation et d'optimisation de la consommation d'applications du TDSI pour la conception des systèmes reconfigurables de type FPGA. Thèse, MESR, 06/12/2006, IETR-Rennes/LESTER.

[FEB06] Jean-Marie Favre, J. Estublier et Mireille Blay-Fornarino. L'ingénierie dirigée par les modèles au-déla du MDA. Hermès Science, Lavoisier, Janvier 2006.

[GHO03] F.Ghozzi. Optimisation d'une bibliothèque de modules matériels de traitement d'images. Conception et test VHDL, implémentation sous forme FPGA. Thèse à l'Université de Bordeaux, 2003.

[GRJ04] Grasset, F. Rousseau, A. A. Jerraya, "Network Interface Generation for MPSoC: from Communication Service Requirements to RTL Implementation", 15th IEEE International Workshop on Rapid System Prototyping (RSP 2004), Geneva, Switzerland, June 2004.

[HAM05] Hamabe, M.: Systemc with mpi for clustering simulation, www5a.biglobe.ne.jp/~hamabe/SystemC

[HeMa00] T.A. Henzinger and R. Majumdar. Symbolic model checking for rectangular hybrid systems. In S. Graf, editor, TACAS 2000: Tools and algorithms for the construction and analysis of systems, Lecture Notes in Computer Science, New-York, 2000. Springer-Verlag.

[HEN99] J. Henkel. A low power hardware/software partitioning approach for core-based embedded systems. In The 36th Design automation conference, 1999.

[IKH01] C. Im, H. Kim, and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers, 2001.

[JER02] Ahmed Jerraya, "Conception de haut niveau des systèmes monopuces", Hermès Science Publication, Lavoisier, 2002

[JER07] A. Jerraya, HW/SW Implementation from Abstract Architecture Models, EDA Consortium San Jose, CA, USA, Pages: 1470 - 1471, 2007.

[JLS02a] N. Julien, J. Laurent, E. Senn, and E. Martin. Power consumption modeling and characterization of the TI C6201. IEEE Micro, 23(5), 2003.

[JLS02b] N. Julien, J. Laurent, E. Senn, and E. Martin. Power Estimation of a C Algorithm Based on the Functional-Level Power Analysis of a Digital Signal Processor. In ISHPC '02 : Proceedings of the 4th International Symposium on High Performance Computing, London, UK, 2002.

[KAV08] A. Koudri, D. Aulagnier, D. Vojtisek, P. Soulard, Using MARTE in a Co-Design Methodology, DATE2008 Friday Workshop - 14th March, 2008.

[KMD05] A. Koudri, S. Meftali, J.-L. Dekeyser. IP integration in embedded systems modeling, in: IP-SOC 2005, IP Based SoC Design Conference, Grenoble, France, December 2005.

[KoAl03] Cindy Kong and Perry Alexander. The Rosetta meta-model framework. In *ECBS*, pages 133–140. IEEE Computer Society, 2003.

[LEB07a] Le Beux et al. A Model Driven Engineering Design Flow to generate VHDL. In International ModEasy'07 Workshop, 2007.

[LEB07b] S. Le Beux, Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles, Thèse. Université des Sciences et Technologies de Lille, Décembre 2007.

[LiHe98] Y. Li and J. Henkel. A framework for estimating and minimizing energy dissipation of embedded hw/sw systems. In The 35th Design automation conference, 1998.

[LJS04] J. Laurent, N. Julien, E. Senn, and E. Martin. Functional level power analysis : An efficient approach for modeling the power consumption of complex processors. In DATE'04 : Proceedings of the conference on Design, automation and test in Europe, DC, USA, 2004.

[LLF01] Loïc Lagadec, Dominique Lavenier, Erwan Fabiani, and Bernard Pottier. Placing, routing, and editing virtual FPGAs. In 11th International Conference on Field-Programmable Logic and Applications, FPL'01, pages 357–366, Belfast, Northern Ireland, UK, août 2001.

[LlGo] R. P. Llopis and K. Goossens. The petrol approach to high-level power estimation. In ISLPED '98 : Proceedings of the 1998 international symposium on Low power electronics and design, CA, USA, 1998.

[LSI04] M. Graphics. Lsim power analyst : Transistor-level simulation, 2004. http://www.mentor.com/lsmpoweranalyst/datasheet.html.

[MCV05] Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. A taxonomy of model transformations. In Jean Bezivin and Reiko Heckel, editors, Language Engineering for Model- Driven Software Development, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.

[MDC05] S. Meftali, A. Dziri, L. Charest, P. Marquet, J.-L. Dekeyser. Soap based distributed simulation environment for system-on-chip (soc) design, in: FDL'05, Lausanne, Switzerland, September 2005.

[MDS05] S. Meftali, J.-L. Dekeyser, Isaac D. Scherson. Scalable Multistage Networks for Multiprocessor System-on-Chip Design, in: International Symposium on Parallel Architectures, Algorithms, and Networks, Las Vegas, Nevada, USA, December 2005.

[MeDe04a] Samy Meftali, Jean-Luc Dekeyser, SoC P2P: A Peer-to-Peer IP Based SoCs Design and Simulation Tool. PRO-VE'04 : 5th IFIP Working Conference on VIRTUAL ENTERPRISES. August, 2004. Toulouse- France.

[MeDe04b] Samy Meftali, Jean-Luc Dekeyser, SoC P2P: A Peer-to-Peer IP Based SoCs Design and Simulation Tool. Virtual Entreprises and Collaborative Networks. Editions Kluwer Academic Pulishers. August, 2004. Toulouse- France

[MeDe04c] Samy Meftali, Jean-Luc Dekeyser, An Optimal Charge Balancing Model for Fast Distributed SystemC Simulation in IP/SoC Design. The 4th IEEE International Workshop : System-on-Chip for Real-Time Applications. July 19 - July 21, 2004. Banff, Alberta - Canada.

[MEF02] S. Meftali. Exploration d'architectures et allocation/affectation mémoire dans les systèmes multiprocesseurs monopuce. Rapport de thèse, Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs, Université de Joseph Fourier, 2002.

[MOD03] Model Technology. ModelSim, Tutorial, Version 5.8. December 16, 2003.

[MSD04] Samy Meftali, Mickael Samyn, Jean-Luc Dekeyser. Approche MDA, avec plateforme SystemC, pour la conception de systèmes monopuces dédiés au traitement de signal intensif. CISC'04. September 2004. Jijel, Algérie.

[MVD03a] Samy Meftali, Joel Vennin, Jean-Luc Dekeyser, A Fast SystemC Simulation Methodology for Multi-level IP/SoC Design , International Workshop on IP Based SoC design, Grenoble, France, November 2003.

[MVD03b] Samy Meftali, Joel Vennin, Jean-Luc Dekeyser, Automatic Generation of, Geographically Distributed, SystemC Simulation Models for IP/SoC Design, 46th IEEE International MWSCAS, Cairo, Egypt, December 2003.

[MVD04] Samy Meftali, Joel Vennin, Jean-Luc Dekeyser. Méthodologie de simulation multi niveaux, pour la conception de systèmes monopuces en SystemC. CISC'04. September 2004. Jijel, Algérie.

[MVD06] S. Meftali, J. Vennin, J.-L. Dekeyser. "An optimized distributed simulation environment for SoC design", in: Annals for Micro and Nano Systems, 2006

[NAB08] Bilel Neji, Yassine Aydi, Rabie Ben atitallah, Samy Meftali, Mohamed Abid, and Jean Luc Dekeyser. "MULTISTAGE INTERCONNECTION NETWORK FOR MPSOC: PERFORMANCES STUDY AND PROTOTYPING ON FPGA". The 3rd International Design and Test Workshop, December 2008 Monastir, Tunisia.

[NIC02] Gabriela Nicolescu. Specification and validation for heterogeneous embedded systems. PhD thesis, INPG, Nov 2002.

[NiMe04] Smail Niar, Samy Meftali. Power Consumption Aware in Cache Memory Design with SystemC. The 16 th International Conference on Microelectronics. Tunis, Tunisia. December 2004.

[OCP08] Open Core Protocol Specification Release 2.2, http://www.ocpip.org/spec_download, 2008.

[OMG03] Object Management Group, Inc., editor. UML2 Infrastructure (Final Adopted Specifcation). Septembre 2003.

[OMG05] Object Management Group, Inc. UML profile for Schedulability, Performance and Time (SPT), Version 1.1.Technical Report formal/05-01-02, OMG, 2005.

[OMG07] Object Management Group. MOF Query / Views / Transformations, version 1.0Beta2, ptc/07-07-07, 2007

[OMG09] Object Management Group, Inc. UML Profile for MARTE, Beta 3. OMG Document Number: ptc/2009-05-13

[PAN07] P. R. Panda. Designing Embedded Processors, A Low Power Perspective, chapter Power Optimization Strategies Targeting the Memory Subsystem, pages 131 – 155. Springer, 2007.

[PBM08] É. Piel, R. Ben Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.-L. Dekeyser, P. 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

[PLK03] Chang Hee Pyoun, Chi Ho Lin, Hi Seok Kim, Jong Wha Chong, The efficient bus arbitration scheme in SoC Environment. iwsoc, pp.311, The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03), 2003.

[PoGa] Synopsys. Power-Gate(TM) : a dynamic, simulation-based, gate-level power analysis tool. Synopsys,http://www.synopsys.com/news/pubs/rsvp/fall97/rsvp_fall97_5.html.

[POW] Synopsys. NanoSim datasheet (timemill and powermill) : Memory and mixed signal verification. http://www.synopsys.com/products/mixedsignal/nanosim/nanosim_ds.pdf.

[QAD10] Imran Rafiq Quadri, "MARTE based Model driven design methodology for targeting dynamically reconfigurable FPGA based SoCs", Thèse de l'univérsité de Lille 1, France, avril 2010.

[QBM08] Imran Rafiq. Quadri, P. Boulet, S. Meftali, J.-L. Dekeyser. Using An MDE Approach for Modeling of Interconnection networks, in: The International Symposium on Parallel Architectures, Algorithms and Networks Conference (ISPAN 08), Sydney, Australia, May 2008

[QEM09] Imran Rafiq Quadri, Yassin Elhillali, Samy Meftali and Jean-Luc Dekeyser "Model based design flow for implementing an Anti-Collision Radar system", 9th International IEEE Conference on ITS Telecommunications (ITS-T 2009), Lille - France, October 2009.

[QMD07] R. Quadri, S. Meftali, J.-L. Dekeyser. An MDE Approach for Implementing Partial Dynamic Reconfiguration in FPGAs, in: 16th International Workshop on IP Based System-on-chip, IP'07, Grenoble, France, December 2007

[QMD08a] Imran Rafiq. Quadri, S. Meftali, J.-L. Dekeyser. A MDE design flow for implementing Partially Dynamically Reconfigurable FPGAs, in: 2nd Colloque Nationale of GDR SOC-SIP, Paris, France, June 2008

[QMD08b] Imran Rafiq. Quadri, S. Meftali, J.-L. Dekeyser. High Level Modeling of Partially Dynamically Reconfigurable FPGAs based on MDE and MARTE, in:Reconfigurable Communication-centric SoCs (ReCoSoC'08), Barcelona, Spain, July 2008

[QMD08c] Imran Rafiq. Quadri, S. Meftali, J.-L. Dekeyser. MARTE based modeling approach for Partial Dynamic Reconfigurable FPGAs, in: Sixth IEEE Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia 2008), Atlanta, USA,

[QMD09a] Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser "From MARTE to dynamically reconfigurable FPGAs : Introduction of a control extension in a model based design flow", Research Report RR-6862, INRIA, March 2009.

[QMD09b] Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser "Integrating Mode Automata Control Models in SoC Co-Design for Dynamically Reconfigurable FPGAs", International Conference on Design and Architectures for Signal and Image Processing (DASIP 09), Nice - France, September 2009.

[QMD09c] Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser "A Model Driven design flow for FPGAs supporting Partial Reconfiguration", International Journal of Reconfigurable Computing, Hindawi Publishing Corporation, June 2009.

[QMD09d] I. Quadri, S. Meftali and J.-L. Dekeyser, High level modeling of dynamic reconfigurable FPGAs, International Journal of Reconfigurable Computing, 2009.

[QMD09e] Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser "MARTE based design approach for targeting Reconfigurable Architectures", 2nd Embedded Systems Conference - ESC'09, Alger - Algeria, May 2009.

[QMD10] Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser "From MARTE to Reconfigurable NoCs: A model driven design methodology", Chapter in Dynamic Reconfigurable Network-on-

Chip Design: Innovations for Computational Processing and Communication: IGI-Global, Tentative publication date: 2010.

[QMM09] Imran Rafiq Quadri, Alexis Muller, Samy Meftali and Jean-Luc Dekeyser "MARTE based design flow for partially reconfigurable Systems-on-Chips", 17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09), Florianapolis - Brazil, October 2009.

[QYG10] Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Eric Rutten, Samy Meftali and Jean-Luc Dekeyser "Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation", Special Issue on Reconfigurable and Multicore Embedded Systems, International Journal of Embedded Systems (IJES), InderScience Publishers March 2010.

[RaWo87] Peter J. Ramadge and W. Murray Wonham. Supervisory Control of a Class of Discrete Event Processes. SIAM Journal on Control and Optimization 25(1), pages 206-230, 1987

[REV08] S.Revol, Profil UML pour TLM: contribution à la formalisation et à l'automatisation du flot de conception et vérification des systèmes-sur-puce, thèse, Institut National Polytechnique de Grenoble, 2008.

[SaCa01] Salefski, B. & Caglar, L., Reconfigurable computing in wireless, *in* 'Proceedings of the 38th annual Design Automation Conference (DAC'01)', ACM, pp. 178–183. 2001.

[SAM05] Mickaël Samyn. Une simulation fonctionnelle d'un système monopuce dédié au traitement du signal intensif - Une approche dirigée par les modèles. Thèse de doctorat, université de Lille 1, France, December 2005

[SBA05] Sedcole, P., Blodget, B., Anderson, J., Lysaght, P. & Becker, T., Modular Partial Reconfiguration in Virtex FPGAs, *in* 'FPL'05', pp. 211–216. 2005.

[SCJ00] A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In Proceedings of the 38th DAC Conference, 2000.

[SeGr02] L. SEMERIA et A. GHOSH. « Methodology for Hardware/Software Co-verification in C/C++ ». Dans *Proceedings of the Asia South Pacific Design Automation Conference*, jan 2002.

[SeKo03] S. Sendall and W. Kozaczynski, 'Model Transformation: The Heart and Soul ofModel-Driven Software Development', *IEEE Software* **20**(5), 42–45. 2003.

[SiBo07] Love Singhal and Elaheh Bozorgzadeh. Novel multi-layer floorplanning for heterogeneous fpgas. In 17th International Conference on Field-Programmable Logic and Applications,FPL'07, pages 613–616, Amsterdman, Netherlands, août 2007.

[SiMu01] Siegmund, R. Muller, D. SystemC^{SV}: an extension of SystemC for mixed multi-level communication modeling and interface-based system design. Design Automation and Test in Europe, DATE 2001.

[SKH08] Schuck, C., Kuhnle, M., Hubner, M. & Becker, J., A framework for dynamic 2D placement on FPGAs, *in* 'IPDPS 2008'. 2008.

[SKM01] Schafer, T., Knapp, A. & Merz, S., Model Checking UML State Machines and Collaborations, *in* 'CAV Workshop on Software Model Checking', ENTCS 55(3). 2001.

[SLJ04] E. Senn, J. Laurent, N. Julien, and E. Martin. SoftExplorer : estimation, characterization and optimization of the power and energy consumption at the algorithmic level. In IEEE PATMOS, Santorin, Grèce, 2004.

[SMD04] Mickael Samyn, Samy Meftali, Jean-Luc Dekeyser, MDA Based, SystemC Code Generation, Applied to Intensive Signal Processing Applications. FDL'04. September 2004. Lille, France

[SMN04] Mickael Samyn, Samy Meftali, Smail Niar, Jean-Luc Dekeyser. Performances Estimation Metamodel for MDA Based SoC Design. International Workshop on IP Based SoC design, Grenoble, France, December 2004.

[SOC06] S. project. An open platform for modelling and simulation of multi-processors system on chip. http://soclib.lip6.fr/Home.html.

[SPA] U. of Berkeley (USA). Spice manual. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

[SPE09] SPEEDS!, 'SPEculative and Exploratory Design in Systems Engineering'. http://www.speeds.eu.com/. 2009.

[TAH08] S.Taha, Modélisation conjointe logiciel/matériel de systèmes temps réel, rapport de thèse, 2008.

[Tai09] J.Taillard, Une approche orientée modèle pour la parallélisation d'un code de calcul éléments finis, thèse, Université des Sciences et Technologies de Lille. Février 2009.

[TMB10] Chiraz Trabelsi, Samy Meftali, Rabie Ben-Atitallah, Jean-Luc Dekeyser, Smail Niar, An MDE Approach for Energy Consumption Estimation in MPSoC Design, RAPIDO 2010, January 2010, Pisa, Italy

[TMN04] Emilian Turbatu, Samy Meftali, Smail Niar, Jean-Luc Dekeyser, An automatic communication synthesis for high level SOC design using transaction level modeling, FDL'04. September 2004. Lille, France

[BED09] Lahbib Bedouj, Estimation de performances dans les FPGAs dynamiquement reconfigurables. Mémoire de master recherche, université de Tanger 2009.

[YSM09] Ysmal Arnaud. Implémentation et évaluation d'un système de surveillance distribué, sur FPGAs. Mémoire de master recherche, université de Lille 1, 2009.

[Tay06] Imen Tayari. Reconfiguration dynamique dans les réseaux multiétages. Mémoire de master recherche, université de Lille 1, 2006.

[Nej08] Bilel Neji. Implémentation de réseaux multiétages sur FPGA. Mémoire de master recherche, ENIS Sfax, 2008.

[TUR04] Emilian Turbatu. Synthèse de la communication dans un environnement basé sur SystemC. Mémoire de master recherche, université de Lille 1. 2004

[VEN03] Joel Vennin. Simulation multi niveaux en SystemC. Mémoire de master recherche, université de Lille 1. 2003

[TMP07] Tumeo, A., Monchiero, M., Palermo, G., Ferrandi, F. & Sciuto, D., 'A Self-Reconfigurable Implementation of the JPEG Encoder', *ASAP 2007* pp. 24–29. 2007.

[TMW94] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software : A first step towards software power minimization. In Transactions on VLSI Systems, 1994.

[TRA04] Trams, M.: Conservative distributed discrete event simulation with systemc using explicit look ahead. Technical report, Digital Force (2004), http://www.digital-force.net

[TRA09] Chiraz Trabelsi "Estimation et évaluation de performance lors de la conception des systèmes mixtes HW/SW", Mémoire de master de l'INSAT Tunis, janvier 209.

[TRC91] Hubert Tardieu, Arnold Rochfeld, and René Colletti. La Méthode Merise : Principes et outils. Editions d'Organisation, 1991.

[VCI01] VSI AllianceVirtual Component Interface Standard Version 2, http://comelec.enst.fr/dessin/canex/VCI.pdf, 2001

[VIN09] Vineet Aggarwal, National Instruments, magazine RTC, avril 2009.

[VLG09] Vidal, J., Lamotte, F. D. & Gogniat, G., A co-design approach for embedded system modeling and code generation with UML and MARTE, *in* 'Design, Automation and Test in Europe (DATE'09). 2009.

[VMD04] Joel Vennin, Samy Meftali, Jean-Luc Dekeyser. Understanding and Extending SystemC User Thread Package to IA-64 Platform. International Workshop on IP Based SoC design, Grenoble, France, December 2004.

[VPC05] J. Vennin, S. Penain, L. Charest, S. Meftali, J.-L. Dekeyser. Embedded scripting inside systemc, in: FDL'05, Lausanne, Switzerland, september 2005.

[WAH07] S. Wood, D. Akehurst, W. Howells, and K. McDonald-Maier. Mapping the design of repetitive structures onto vhdl. In *International ModEasy¿07 Workshop*, 2007.

[XIL04] Xilinx. Software Manual and Documentation for ISE6.1. Technical Document 2004.

[XIL06] Xilinx, Early Access Partial Reconfigurable Flow. http://www.xilinx.com/support/prealounge/protected/index.htm. 2006.

[XIL07] Virtex-II Pro and Virtex-II Pro X FPGA User Guide. UG012 (v4.2) 5 November 2007

[YBM02] L. Yung-Hsiang, L. Benini, and G. D. Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1284–1305, 2002.

[YU08a] Yu, H., A MARTE-Based Reactive Model for Data-Parallel Intensive Processing: Transformation toward the Synchronous Model, PhD thesis, USTL/LIFL, France. 2008.

[YU08b] Yu, H., Gamatié, A., Rutten, E. & Dekeyser, J.-L., 'Safe Design of High-Performance Embedded Systems in a MDE. framework', *Innovations in Systems and Software Engineering (ISSE)* **4**(3), 215–222. 2008.

[YVK00] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin. The Design and Use of Simple-Power : A Cycle Accurate Energy Estimation Tool. In Design Automation Conf, June 2000.

Annexe 1: ACTIVITÉS POUR LA PERIODE (2002-2010)

Civilité : **Mr** Nom patronymique : **Meftali** Nom usuel : Prénom : **Samy**

Date de naissance : 5 novembre 1975
Grade : Maître de conférences
Etablissement d'affectation : Université de Lille 1 (USTL)
Section de CNU : 27
Unité de recherche d'appartenance : LIFL (Laboratoire d'informatique fondamentale de Lille) et INRIA Lille Nord Europe

1. Grade et poste

- Maître de Conférences classe normale, Section Informatique (27ième Section) USTL, Université de Lille 1, LAMIH LIFL (Laboratoire d'informatique fondamentale de Lille), équipe DaRT. Responsable : Pr. Jean-Luc Dekeyser. Date de nomination dans le poste octobre 2004.

- Chercheur à l'INRIA Lille- Nord Europe dans l'EPI DART, depuis septembre 2005. Responsable Pr. J-L Dekeyser.

2. Titres universitaires

- Doctorat de l'université de Grenoble I en Informatique, Sujet : "Allocation et affectation mémoire dans les systèmes multiprocesseurs monopuces", Laboratoire TIMA de Grenoble. Direction Ahmed A. Jerraya. Soutenue septembre 2002.

- Diplôme d'Etudes Approfondies (DEA), Institut National Polytechnique de Grenoble (INPG – ENSIMAG) université de Grenoble 1, Laboratoire GILCO de Lille, soutenu en juin 1999.

3. Publications et production scientifique

3.2. Revues internationales avec comités de sélection, et chapitres de livres

1. Imran Rafiq Quadri, Majdi Elhaji, **Samy Meftali** and Jean-Luc Dekeyser*:* From MARTE to Reconfigurable NoCs: A model driven design methodology. Chapter VI in Dynamic Reconfigurable Network-on-Chip Design: Innovations for Computational Processing and Communication. Publisher: Information Science Reference. Avril 2010.
2. Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Eric Rutten, **Samy Meftali** and Jean-Luc Dekeyser "Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation", Special Issue on Reconfigurable and Multicore Embedded Systems, International Journal of Embedded Systems (IJES), InderScience Publishers , Tentative publication date : March 2010.
3. Imran Rafiq Quadri, **Samy Meftali** and Jean-Luc Dekeyser "From MARTE to Reconfigurable NoCs: A model driven design methodology", Chapter in Dynamic

Reconfigurable Network-on-Chip Design: Innovations for Computational Processing and Communication : IGI-Global, Tentative publication date: 2010.
4. Jean-Luc Dekeyser, Abdoulaye Gamatié, **Samy Meftali** and Imran Rafiq Quadri (In alphabetical order) "High-level modeling of dynamically reconfigurable heterogeneous systems", Chapter 7 in Heterogeneous Embedded Systems - Design Theory and Practice : Springer, Tentative publication date: 2010.
5. Imran Rafiq Quadri, **Samy Meftali** and Jean-Luc Dekeyser "A Model Driven design flow for FPGAs supporting Partial Reconfiguration", *International Journal of Reconfigurable Computing*, Hindawi Publishing Corporation, June 2009.
6. **S. Meftali**, J. Vennin, J.-L. Dekeyser. "An optimized distributed simulation environment for SoC design", in: Annals for Micro and Nano Systems, 2006
7. **Samy Meftali**, Jean-Luc Dekeyser, SoC P2P: A Peer-to-Peer IP Based SoCs Design and Simulation Tool. Virtual Entreprises and Collaborative Networks. Editions Kluwer Academic Pulishers. August, 2004. Toulouse- France

3.2. Conférences internationales avec actes et comités de sélection

1. M. Elhaji, P. Boulet, S. Meftali, A.Zitouni, J.Dekeyser & R. Tourki. An MDE approach for modeling network on chip topologies. 5th INTERNATONAL CONFERENCE ON Design & Technology of Integrated Systems in Nanoscale Era. March 2010 Hammamet, TUNISIA.
2. Chiraz Trabelsi, **Samy Meftali**, Rabie Ben-Atitallah, Jean-Luc Dekeyser, Smail Niar, An MDE Approach for Energy Consumption Estimation in MPSoC Design, RAPIDO 2010, January 2010, Pisa, Italy
3. Imran Rafiq Quadri, Alexis Muller, **Samy Meftali** and Jean-Luc Dekeyser "MARTE based design flow for partially reconfigurable Systems-on-Chips", *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)*, Florianapolis - Brazil, October 2009.
4. Imran Rafiq Quadri, Yassin Elhillali, **Samy Meftali** and Jean-Luc Dekeyser "Model based design flow for implementing an Anti-Collision Radar system", 9th International IEEE Conference on ITS Telecommunications (ITS-T 2009), Lille - France, October 2009.
5. Imran Rafiq Quadri, **Samy Meftali** and Jean-Luc Dekeyser "Integrating Mode Automata Control Models in SoC Co-Design for Dynamically Reconfigurable FPGAs", International Conference on Design and Architectures for Signal and Image Processing (DASIP 09), Nice - France, September 2009.
6. É. Piel, R. Ben Atitallah, P. Marquet, **S. Meftali**, S. Niar, A. Etien, J.-L. Dekeyser, P. 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
7. Imran Rafiq. Quadri, P. Boulet, **S. Meftali**, J.-L. Dekeyser. Using An MDE Approach for Modeling of Interconnection networks, in: The International Symposium on Parallel Architectures, Algorithms and Networks Conference (ISPAN 08), Sydney, Australia, May 2008
8. Imran Rafiq. Quadri, **S. Meftali**, J.-L. Dekeyser. A MDE design flow for implementing Partially Dynamically Reconfigurable FPGAs, in: 2nd Colloque Nationale of GDR SOC-SIP, Paris, France, June 2008
9. Imran Rafiq. Quadri, **S. Meftali**, J.-L. Dekeyser. High Level Modeling of Partially Dynamically Reconfigurable FPGAs based on MDE and MARTE, in:Reconfigurable Communication-centric SoCs (ReCoSoC'08), Barcelona, Spain, July 2008

10. Bilel Neji, Yassine Aydi, Rabie Ben atitallah, **Samy Meftali**, Mohamed Abid, and Jean Luc Dekeyser. "MULTISTAGE INTERCONNECTION NETWORK FOR MPSOC: PERFORMANCES STUDY AND PROTOTYPING ON FPGA". The 3rd International Design and Test Workshop, December 2008 Monastir, Tunisia.
11. Imran Rafiq. Quadri, **S. Meftali**, J.-L. Dekeyser. MARTE based modeling approach for Partial Dynamic Reconfigurable FPGAs, in: Sixth IEEE Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia 2008), Atlanta, USA,
12. Yassine Aydi, **Samy Meftali**, Mohamed Abid, Jean-Luc Dekeyser. Design and Performance Evaluation of a Reconfigurable Delta MIN for MPSOC. In 19th International Conference on Microelectronics (ICM'07), Cairo, Egypt, December 2007.
13. Y. Aydi, **S. Meftali**, M. Abid, J.-L. Dekeyser. Dynamicity Analysis of Delta MINs for MPSoC Architectures, in: Conférence internationale des Sciences et Techniques de l'Automatique (STA'07), November 2007
14. P. Marquet, S. Duquennoy, S. Le Beux, S. Meftali, J.-L. Dekeyser. *Massively Parallel Processing on a Chip*, in: *ACM Int'l Conf. on Computing Frontiers, Ischia, Italy*, May 2007
15. R. Quadri, **S. Meftali**, J.-L. Dekeyser. An MDE Approach for Implementing Partial Dynamic Reconfiguration in FPGAs, in: 16th International Workshop on IP Based System-on-chip, IP'07, Grenoble, France, December 2007
16. R. Ben Atitallah, S. Niar, S. Meftali, J.-L. Dekeyser. An MPSoC performance estimation framework using transaction level modeling, in: IEEE RTCSA'2007, Daegu, Korea, 2007
17. R. B. Atitallah, S. Niar, A. Greiner, **S. Meftali**, J. L. Dekeyser. Estimating Energy Consumption for an MPSoC Architectural Exploration, in: ARCS'06: Architecture of Computing Systems, Frankfurt, Germany, March 2006. http://arcs06.cs.uni-frankfurt.de/
18. R. B. ATITALLAH, L. BONDE, S. NIAR, **S. MEFTALI**, J.-L. DEKEYSER. Multilevel MPSoC performance evaluation using MDE approach., in: International Symposium on System-on-Chip 2006, Tampere, Finland, november 2006
19. R. B. ATITALLAH, S. NIAR, **S. MEFTALI**, J.-L. DEKEYSER. Accelerating MPSoC Performance Evaluation with TLM., in: "Advanced Computer Architecture and compilation for Embedded Systems ACACES", HIPEAC network of Excellence summer school, Laquilia, Italy, ISBN 90 382 0981 9, july 2006, p. 229–232
20. S. DUQUENNOY, S. LE BEUX, P. MARQUET, **S. MEFTALI**, J. DEKEYSER. MpNoC Design: Modeling and Simulation, in: 15th edition of IP/SOC, 2006
21. J.-L. DEKEYSER, P. MARQUET, **S. MEFTALI**, C. DUMOULIN, P. BOULET, S. NIAR. Why to do without Model Driven Architecture in embedded system codesign?, in: first annual IEEE BENELUX/DSP Valley Signal Processing Symposium, SPS-DARTS 2005, Antwerp, Belgium, April 2005. http://acivs.org/sps2005
22. KOUDRI, **S. MEFTALI**, J.-L. DEKEYSER. IP integration in embedded systems modeling, in: IP-SOC 2005, IP Based SoC Design Conference, Grenoble, France, December 2005. .
23. **S. Meftali**, J.-L. Dekeyser, Isaac D. Scherson. Scalable Multistage Networks for Multiprocessor System-on-Chip Design, *in:* International Symposium on Parallel Architectures, Algorithms, and Networks, Las Vegas, Nevada, *USA*, December 2005.
24. Koudri, **S. Meftali**, J.-L. Dekeyser. *IP* integration in embedded systems modeling, *in:* IP-SOC 2005, IP Based SoC Design Conference, Grenoble, France, December 2005.
25. J. Vennin, S. Penain, L. Charest, **S. Meftali**, J.-L. Dekeyser. Embedded scripting inside systemc, *in:* FDL'05, Lausanne, Switzerland, september 2005.
26. **S. Meftali**, A. Dziri, L. Charest, P. Marquet, J.-L. Dekeyser. Soap based distributed simulation environment for system-on-chip (soc) design, *in:* FDL'05, Lausanne, Switzerland, September 2005.
27. Jean-Luc Dekeyser, Philippe Marquet, **Samy Meftali**, Cédric Dumoulin, Pierre Boulet,

and Smail Niar. Why to do without Model Driven Architecture in embedded system codesign?. In The first annual IEEE BENELUX/DSP Valley Signal Processing Symposium, SPS-DARTS 2005, Antwerp, Belgium, April 2005.
28. Joel Vennin, **Samy Meftali**, Jean-Luc Dekeyser. Understanding and Extending SystemC User Thread Package to IA-64 Platform. International Workshop on IP Based SoC design, Grenoble, France, December 2004.
29. Mickael Samyn, **Samy Meftali**, Smail Niar, Jean-Luc Dekeyser. Performances Estimation Metamodel for MDA Based SoC Design. International Workshop on IP Based SoC design, Grenoble, France, December 2004.
30. Smail Niar, **Samy Meftali**. Power Consumption Aware in Cache Memory Design with SystemC. The 16 th International Conference on Microelectronics. Tunis, Tunisia. December 2004.
31. Mickael Samyn, **Samy Meftali**, Jean-Luc Dekeyser, MDA Based, SystemC Code Generation, Applied to Intensive Signal Processing Applications. FDL'04. September 2004. Lille, France
32. Emilian Turbatu, **Samy Meftali**, Smail Niar, Jean-Luc Dekeyser, An automatic communication synthesis for high level SOC design using transaction level modeling, FDL'04. September 2004. Lille, France
33. **Samy Meftali**, Jean-Luc Dekeyser, An Optimal Charge Balancing Model for Fast Distributed SystemC Simulation in IP/SoC Design. The 4th IEEE International Workshop : System-on-Chip for Real-Time Applications. July 19 - July 21, 2004. Banff, Alberta - Canada.
34. **Samy Meftali**, Jean-Luc Dekeyser, SoC P2P: A Peer-to-Peer IP Based SoCs Design and Simulation Tool. PRO-VE'04 : 5th IFIP Working Conference on VIRTUAL ENTERPRISES. August, 2004. Toulouse- France.
35. **Samy Meftali**, Joel Vennin, Jean-Luc Dekeyser, Automatic Generation of, Geographically Distributed, SystemC Simulation Models for IP/SoC Design, 46th IEEE International MWSCAS, Cairo, Egypt, December 2003.
36. **Samy Meftali**, Joel Vennin, Jean-Luc Dekeyser, A Fast SystemC Simulation Methodology for Multi-level IP/SoC Design , International Workshop on IP Based SoC design, Grenoble, France, November 2003.

3.3. Autres conférences (posters, conférences nationales

1. Imran Rafiq Quadri, **Samy Meftali** and Jean-Luc Dekeyser "MARTE based design approach for targeting Reconfigurable Architectures", 2nd Embedded Systems Conference - ESC'09, Alger - Algeria, May 2009.
2. Imran Rafiq Quadri, **Samy Meftali** and Jean-Luc Dekeyser "From MARTE to dynamically reconfigurable FPGAs : Introduction of a control extension in a model based design flow", Research Report RR-6862, INRIA, March 2009.
3. **Samy Meftali**, Joel Vennin, Jean-Luc Dekeyser. Méthodologie de simulation multi niveaux, pour la conception de systèmes monopuces en SystemC. CISC'04. September 2004. Jijel, Algérie.
4. **Samy Meftali**, Mickael Samyn, Jean-Luc Dekeyser. Approche MDA, avec plateforme SystemC, pour la conception de systèmes monopuces dédiés au traitement de signal intensif. CISC'04. September 2004. Jijel, Algérie.
5. **Samy Meftali**. Memory Architectures Exploration in MP-SoC Design. PhD Forum at VLSI SoC 2003, Darmstadt, Germany, December 2003.

3.4. Publications pendant la thèse (1999 – 2002)

1. ·GHARSALLI F., MEFTALI S., ROUSSEAU F., JERRAYA A. A. *Automatic generation of embedded memory wrapper generation for multiprocessor SoC*. 39th Design Automation Conference (DAC'02), New Orleans, USA, June 10-14, 2002,
2. ·GHARSALLI F., LYONNARD D., MEFTALI S., ROUSSEAU F., JERRAYA A. A. *Unifying memory and processor wrapper architecture for multiprocessor SoC design* . International Symposium on System Synthesis (ISSS'02), Kyoto, Japan, October 2-4, 2002 ,
3. MEFTALI S., GHARSALLI F., ROUSSEAU F., JERRAYA A. A. Chapitre Automatic code-transformations and architecture refinement, for application-specific SoC du livre Soc Design Methodologies. Editions Kluwer Academic Pulishers, USA, Juillet 2002.
4. Samy Meftali. Shared Memory Architecture Design for Application-Specific Multiprocessor Systems-on-Chip. PhD Forum at DAC 2002, New Orleans, USA, June 2002.
5. MEFTALI S., GHARSALLI F., ROUSSEAU F., JERRAYA A. A. *An optimal memory allocation for application-specific multiprocessor system-on-chip* . 13th International Symposium on System Synthesis (ISSS 2001), Montreal, Canada, September 30 - October 3 2001.
6. ·MEFTALI S., GHARSALLI F., ROUSSEAU F., JERRAYA A. A. *Automatic code-transformations and architecture refinement, for application-specific SoC* . IFIP International Conference on Very Large Scale Integration - The Global System on Chip Design & CAD Conference (VLSI-SOC 2001), Montpellier, France, December 3-5 2001.

4. Encadrement doctoral et scientifique

4.1. Thèses achevées

- Michael Samyn (co-direction 50% avec Jean-Luc Dekeyser)
  - Sujet : Une simulation fonctionnelle d'un système monopuce dédié au traitement du signal intensif - Une approche dirigée par les modèles
  - Début : Octobre 2002
  - Fin : thèse soutenue en décembre 2005
  - Financement : Allocation ministère
- Imran Rafiq Quadri (co-direction 50% avec Jean-Luc Dekeyser)
  - Sujet : Une méthodologie de conception dirigée par les modèles basée sur MARTE pour cibler les systèmes sur puces basés sur les FPGA dynamiquement reconfigurables
  - Début : Octobre 2006
  - Fin : thèse soutenue en avril 2010
  - Financement : Bourse Pakistan
- Rabie Ben Atitallah (co-encadrement 20% avec Jean-Luc Dekeyser et Smail Niar)
  - Sujet : Modèles et simulation des systèmes sur puce multiprocesseurs - Estimation des performances et de la consommation d'énergie
  - Début : janvier 2005
  - Fin : thèse soutenue en 2008
  - Financement : projet Interreg
- Joel Vennin (co-direction 50% avec Jean-Luc Dekeyser)
  - Sujet : Environnement de simulation hétérogène, multi niveaux et distribué
  - Début : Octobre 2003
  - Fin : thèse arrêtée en décembre 2005 à cause du dépôt de bilan de l'entreprise la finançant
  - Financement : thèse CIFRE

4.2. Thèses en cours

- Chiraz Trabelsi (co-direction 50% avec Jean-Luc Dekeyser)
  - Sujet : Contrôle distribué et intelligent dans les FPGA dynamiquement reconfigurables
  - Début : Octobre 20010
  - Financement : bourse de l'INRIA
- Sana Cherif (Co-encadrement 50% avec Jean-Luc Dekeyser et El-Bay Bourennane de l'université de Bourgogne)
  - Sujet : Extension du profil MARTE pour supporter les architectures dynamiquement reconfigurables
  - Début : février 20010
  - Financement : ANR Famous

- Majdi Elhaji (Co-encadrement 30% avec Jean-Luc Dekeyser et Rached Torki)
    - Sujet : Co-Design de l'application H264 et implantation sur un NoC-GALS
    - Début : janvier 2009
    - Financement : MED 3+3
- Yassine Aydi (Co-encadrement 30% avec Jean-Luc Dekeyser et Mohamed Abid)
    - Sujet : Conception et évaluation des systèmes embarqués à base de MINs
    - Début : octobre 2006
    - Fin : soutenance prévue en octobre 2010
    - Financement : inscription et financement en Tunisie

- Un nouveau doctorant sera recruté en septembre 2010 dans le cadre de l'anr Famous. Il sera co dirigé par moi-même et Jean-Philipe Digué (Université de Bretagne Sud)

4.3. Masters achevés

- Sana Cherif (encadrement 100%)
    - Sujet : extension du méta-modèle d'architecture de MARTE: modélisation des FPGAs et prise en compte de leur consommation d'énergie
    - reconfigurables
    - Début : janvier 2009
    - Fin : juin 2009
- Chiraz Trabelsi (encadrement 100%)
    - Sujet : Estimation et évaluation de performance lors de la conception des systèmes mixtes HW/SW
    - Début : janvier 2008
    - Fin : juin 2008
- Lahbib Bedouj (encadrement 100%)
    - Sujet : Estimation de performances dans les FPGAs dynamiquement reconfigurables
    - Début : janvier 2009
    - Fin : juin 2009
- Ysmal Arnaud ((encadrement 100%))
    - Sujet : Implémentation et évaluation d'un système de surveillance distribué, sur FPGAs
    - Début : mars 2009
    - Fin: juin 2009
- Imen Tayari (encadrement 100%)
    - Sujet : Reconfiguration dynamique dans les réseaux multiétages
    - Début : février 2006
    - Fin: juin 2006
- Bilel Neji (encadrement 100%)
    - Sujet : Implémentation de réseaux multiétages sur FPGA
    - Début : mars 2008
    - Fin: juin 2008

- Ali Koudri (encadrement 50%)
    - Sujet : Interconnexion systématique avec des interfaces OCP en SystemC
    - Début : février 2005
    - Fin: juin 2005
- Emilian Turbatu (encadrement 50% avec Smail NIAR)
    - Sujet : Synthèse de la communication dans un environnement basé sur SystemC
    - Début : février 2004
    - Fin: juin 2004
- Joel Vennin ((encadrement 100%))
    - Sujet : Simulation multi niveaux en SystemC
    - Début : février 2003
    - Fin: juin 2003

5. Rayonnement

- Présentation sur invitation à l'école d'hiver Fetch 2009
- Séminaire sur invitation à l'école doctorale en informatique de Béjaia (Algérie) en mai 2009 (cette visite m'a permit de monter un projet AUF avec Béjaia et l'ENIS de Sfax en Tunisie. Ce projet a été soumis en mars 2010, et est en seconde phase d'évaluation, il a été sélectionné parmi 22 projets/144 lors de la première phase d'évaluation).
- Cours sur deux semaines, sur invitation, à la faculté polytechnique d'Asuncion (Paraguay) en 2004, sur la programmation parallèle et les architectures multiprocesseurs avancées.
- Cours sur une semaine, sur invitation, à l'école doctorale d'informatique de l'université de Béjaia (Algérie) en 2007, sur la conception des systèmes embarqués.
- Séminaire sur invitation à l'université d'Oran (Algérie) en mai 2009
- Séminaire sur invitation à l'université de Tanger (Maroc) en décembre 2008
- Membre du comité de programme de DATE « Design Automation ans Test in Europe » (2006 et 2007)
- Membre du comité d'organisation de DSD 2010
- Review pour des journaux (IJCAET, ACM TODAYS, IEEE Computer Architecture Letters …)
- Plusieurs présentations dans les GDR (SoC SIP)
- Membre du vivier de recrutement MdC de l'université de Valenciennes (2009)
- Membre du réseau d'excellence Heapic
- Responsable scientifique et coordinateur de plusieurs projets scientifiques nationaux et internationaux (voir la section responsabilités scientifiques)

6. Responsabilités scientifiques

- Responsable scientifique et coordinateur de l'**ANR Famous** : Flot de modélisation et de conception rapide pour les systèmes dynamiquement reconfigurables

  - Date de début : 1er décembre 2009
  - Durée : 48 mois
  - Aide ANR : 980 873 k€
  - Partenaires : INRIA Lille – Nord Europe (coordinateur)
    - Université de Bretagne Sud
    - Université de Bourgogne
    - INRIA Grenoble
    - Sodius (PME)
  - Résumé : La reconfiguration dynamique est un moyen efficace pour rendre les systèmes flexibles et adaptables à une classe d'applications. Cependant, sa prise en compte à partir d'un haut niveau d'abstraction jusqu'à l'implémentation, n'est pas supportée par les outils de conception actuels.

    Le projet proposé vise à présenter une méthodologie complète qui prend en compte la reconfiguration dynamique du matériel, et propose les mécanismes nécessaires pour exploiter entièrement ces possibilités pendant l'exécution.

    Famous s'intéresse aux modèles de très haut niveau (UML), méthodes de compilation et d'exécution ainsi que les techniques d'analyse et de vérification. L'objectif est de fournir des outils pour une conception de qualité, améliorants la productivité, tout en garantissant l'optimisation des ressources matérielles utilisées et en réduisant le temps de mise sur le marché.

- Responsable et coordinateur du projet INRIA MED 3 +3 (MVAR) : **Modélisation et validation des architectures réseaux sur puce (Networks-on-chip NoC) : Application aux systèmes multimédia (MVAR)**
  - Date de début : 1er janvier 2009
  - Durée : 36 mois
  - Budget : 60 k€
  - Partenaires : INRIA Lille – Nord Europe (coordinateur)
    - Université de Monastir (Tunisie)
    - Institut National d'Informatique (Alger)
    - Université de Las Palmas (Espagne

- Co-responsable d'un projet EGID Ulysse
  - Date de début : 1er janvier 2009
  - Durée : 12 mois
  - Partenaires : INRIA Lille – Nord Europe (coordinateur)
    - Université College of Dubin (UCD Irlande)

- Membre du conseil de l'UFR IEEA depuis 2007

- Responsable de la mention MIAGE de la licence informatique depuis 2007 (participation active à la rédaction de la maquette de l'offre de formation 2010 – 2014 évaluée A+)

7. Participation aux projets nationaux et internationaux

En plus des projets cités dans la section précédente où j'occupe des responsabilités scientifiques et de coordination, j'ai (et je participe) participé aux projets suivants :

- Participation au projet européen InterregIII Modeasy (MOdel Driven dEsign for Automative Savety embedded sYstem) mené par l'équipe INRIA-FUTURS DART, le laboratoire d'Opto-Acousto-Électronique de l'IEMN à l'université de Valenciennes et le département d'électronique de l'université de Kent à Canterbury (UK). Période : 2005 à 2009. Objectif du projet : Conception d'un environnement complet d'évaluation et de simulation des systèmes embarqués depuis la modélisation abstraite UML jusqu'à l'implémentation sous formes de FPGA. Application au radar-anticollision. Mon apport : Co-estimation des performances et de la consommation de puissance du niveau RTL vers le niveau TLM (*Transactional Level Modeling*) de la plateforme embarquée.

- Participation au montage et la coordination du projet STIC INRIA "KSOURS : Modèle, vérification et intégration de MPPSoC sur des architectures reconfigurables". mené par l'équipe INRIA-FUTURS DART, l'ENIS de SFAX (TUNISIE) et l'INSAT de TUNIS (TUNISIE). Période : janvier 2008 à décembre 2010

- Participation au projet ANR OpenPeople (2009 – 2011)

8. Mobilité

- Séjours d'un mois au CECS, Université de Californie à IRVINE, UCI, USA : au cours de ce séjour dans l'équipe de Daniel Gajski, j'ai travaillé sur la co-simulation multi-langage SystemC – SpecC

9. Collaborations internationales

- CECS de l'Université de Californie à Irvine (USA), depuis 2005 sur la conception de NoCs. Contact : Isaac D. Scherson
- UCD (Dublin en Irlande), depuis 2009, sur le datamining distribué dans les SoC. Contact : Tahar Kechadi
- INSAT de Tunis (Tunisie) : depuis 2008 dans le cadre des projets STIC Tunisie, sur la thématique de l'estimation de la consommation d'énergie dans les SoC. Contact : Abdarrazek Jemai.

- ENIS de SFAX (Tunisie) : depuis 2007 dans le cadre des projets STIC Tunisie, sur les architectures à base de réseaux multi-étages et l'implémentation sur FPGAs. Contact : Mohamed Abid.
- Université de Béjaia (Algérie) : depuis 2009 sur des invitations de l'université de Béjaia, puis montage d'un projet AUF, sur les méthodes heuristiques pour l'exploration d'architectures de type MPPSoC. Contact : Abdelkamel Tari.
- INI d'Alger (Algérie) : depuis 2009 dans le cadre du projet euromed 3+3, sur les heuristiques de placement dans la conception de systèmes à base de NoCs. Contact : Mouloud Koudil.
- Espagne

10. Enseignement

- ATER de septembre 2002 à aout 2004 à l'université de Lille1.
- Maître de conférences à l'université de Lille1, depuis octobre septembre 2004.

Mes interventions en termes d'enseignements couvrent les trois cycles de l'enseignement supérieur. Ci-dessous la liste des modules j'ai intervenu. Pour la majorité de ces modules, j'assure le cours et j'anime aussi le module : organisation des TD et des TP, synchronisation avec les autres intervenants de Td et TP, rédactions des sujets d'examens, .etc.

10.4. Troisième cycle

Formation : Master 2 professionnel, parcours Informatique Distribuée, Embarquée et Applications IDEAL (Université de Valenciennes)

Cours : Conception et évaluation des performances des systèmes embarqués (SE) 12h, de 2005 à 2007

- Modélisation des systèmes embarqués en SystemC
- Techniques et niveaux de simulation en SystemC

Formation : Master 2 recherche, parcours SEIGLE (Université de Lille1)

Cours : Conception des systèmes embarqués (SE) 3h, de depuis 2004

- Modélisation et simulation des systèmes embarqués

Formation : Master 2 recherche, parcours SEIGLE (Université de Lille1)

Cours : Systèmes d'exploitation embarqués et temps réel 3h, de depuis 2007

- Implémentation sur FPGA
- La notion de reconfiguration dynamique

10.5. Deuxième cycle

Formation : Master1 informatique (Université de Lille 1)
Cours : Simulation des systèmes et architectures matérielles, 40h depuis 2005.

- Introduction aux architectures embarquées
- Méthodes de simulation de matériel
- SystemC

Formation : Licence 3 (IUP 2 GMI) informatique (Université de Lille 1)

Cours : Architecture des ordinateurs, 36h de 2002 à 2007.

- Introduction aux architectures des ordinateurs
- Programmation assembleur (INTEL 32) et microprogrammation
- Introduction aux mémoires caches

Formation : Licence 3 (MIAGE), informatique (Université de Lille 1)

Cours : Réseaux informatiques, 36h depuis 2007.

- Couches réseaux
- Paradigme client serveur.

Formation : Licence 3 formation initiale et formation continue (MIAGE), informatique (Université de Lille 1)

Cours : Pratique du C, 2 x 36h depuis 2007.

- Syntaxe du langage C
- Appels de fonctions et passage de paramètres
- Gestion de pointeurs et allocation dynamique
- Manipulation des fichiers

Formation : Licence 3 informatique (Université de Lille 1)

Cours : Détermination du projet professionnel, 9h depuis 2010.

- Les métiers de l'informatique
- Techniques de recherche de stage

10.6. Premier cycle

Formation : Licence informatique L2 (Université de Lille 1)

Cours : Introduction à l'architecture des ordinateurs, 15h depuis 2005.

- Algèbre de Bool, logique combinatoire et séquentielle.
- Fonctionnement du microprocesseur
- Introduction aux architectures mémoire

Chapitre 7: Publications représentatives

Chapter 1

MODELS FOR CO-DESIGN OF HETEROGENEOUS DYNAMICALLY RECONFIGURABLE SOCS

Jean-Luc Dekeyser,[1] Abdoulaye Gamatié,[1] Samy Meftali,[1] and Imran Rafiq Quadri[1]

[1]*INRIA Lille Nord Europe - LIFL - USTL - CNRS*
Parc Scientifique de la Haute Borne, Park Plaza - Batiment A
40 avenue Halley,
59650 Villeneuve d'Ascq, FRANCE
{Firstname.LastName}@lifl.fr

Abstract The design of Systems-on-Chip is becoming an increasing difficult challenge due to the continuous exponential evolution of the targeted complex architectures and applications. Thus, seamless methodologies and tools are required to resolve the SoC design issues. This chapter presents a high level component based approach for expressing system reconfigurability in SoC co-design. A generic model of reactive control is presented for Gaspard2, a SoC co-design framework. Control integration in different levels of the framework is explored along with a comparison of their advantages and disadvantages. Afterwards, control integration at another high abstraction level is investigated which proves to be more beneficial then the other alternatives. This integration allows to integrate reconfigurability features in modern SoCs. Finally a case study is presented for validation purposes. The presented works are based on Model-Driven Engineering (MDE) and UML MARTE profile for modeling and analysis of real-time embedded systems.

Keywords: **SoC co-design, Component based approach, System adaptivity, Reactive control, Model-Driven Engineering, MARTE, UML, FPGAs, Partial dynamic reconfiguration**

1. Introduction

Since the early 2000s, Systems-on-chip (or SoCs) have emerged as a new paradigm for embedded systems design. In a SoC, the computing units: programmable processors; memories, I/O devices, etc., are all integrated into a single chip. Moreover, multiple processors can be integrated into a SoC (Multiprocessor System-on-Chip, MPSoC) in which the communication can be achieved through Networks-on-Chips (NoCs). Some examples of domains where SoCs are used are: multimedia, automotive, defense and medical applications.

SoC complexity and need of reconfiguration

As the computational power increases for SoCs, more functionalities are expected to be integrated in these systems. As a result, more complex software applications and hardware architectures are integrated, leading to a *system complexity* issue which is one of the main hurdles faced by designers. The fallout of this complexity is that the system design, particularly software design, does not evolve at the same pace as that of hardware. This has become a critical issue and has finally led to the *productivity gap*.

Reconfigurability is also a critical issue for SoCs which must be able to cope with end user environment and requirements. For instance, mode-based control plays an important role in multimedia embedded systems by allowing to describe Quality-of-Service (QoS) choices: 1) changes in executing functionalities, e.g., color or black and white picture modes for modern digital cameras; 2) changes due to resource constraints of targeted platforms, for instance switching from a high memory consumption mode to a smaller one; or 3) changes due to other environmental and platform criteria such as communication quality and energy consumption. A suitable control model must be generic enough to be applied to both software and hardware design aspects.

The reduction in complexity of SoCs, while integrating mechanisms of system reconfiguration in order to benefit from QoS criteria, offers an interesting challenge. Several solutions are presented below.

Component based design

An effective solution to SoC co-design problem consists in raising the design abstraction levels. This solution can be seen through a *top-down* approach. The important requirement is to find efficient design methodologies that raise the design abstraction levels to reduce overall SoC

complexity. They should also be able to express the control to integrate reconfigurability features in modern embedded systems.

Component based design is also a promising alternative. This approach increases productivity of software developers by reducing the amount of efforts needed to develop and maintain complex systems [E. 03]. It offers two main benefits. First, it offers an incremental or *bottom-up* system design approach permitting to create complex systems, while making system verification and maintenance more tractable. Secondly, this approach allows reuse of development efforts as component can be re-utilized across different software products.

Controlling system reconfiguration in SoCs can be expressed via different component models. Automata based control is seen as promising as it incorporates aspects of modularity that is present in component based approaches. Once a suitable control model is chosen, implementation of these reconfigurable SoC systems can be carried out via Field Programmable Gate Arrays (or FPGAs). FPGAs are inherently reconfigurable in nature. State of the art FPGAs can change their functionality at *runtime*, known as Partial Dynamic Reconfiguration (PDR) [P. 06]. These FPGAs also support internal self dynamic reconfiguration, in which an internal controller (a *hardcore/softcore* embedded processor) manages the reconfiguration aspects.

Finally the usage of *high level component based design approach* in development of real-time embedded systems is also increasing to address the compatibility issues related to SoC co-design. High abstraction level SoC co-modeling design approaches have been developed in this context, such as Model-Driven Engineering (MDE) [OMG07] that specify the system using the UML graphical language. MDE enables high level system modeling (of both software and hardware), with the possibility of integrating heterogeneous components into the system. *Model transformations* [T. 06] can be carried out to generate executable models from high level models. MDE is supported by several standards and tools.

Our contribution relates to the proposal of a high level component based SoC co-design framework which has been integrated with suitable control models for expressing reconfigurability. The control models are first explored at different system design levels along with a brief comparison. Afterwards, the control model is explored at another design abstraction level that permits to link the system components with respective implementations. This control model proves more beneficial as it allows to exploit reconfigurability features in SoC by means of partial dynamic reconfiguration in FPGAs. Finally a case study is illustrated which validates our design methodology.

The plan of the chapter is as follows. Section 2 gives an overview of some related works while section 3 defines the notions associated with component based approaches. Section 4 introduces our SoC co-design framework while section 5 illustrates a reactive control model. Section 6 compares control models at different levels in our framework. Section 7 provides a more beneficial control model in our framework, illustrated with a case study. Finally section 8 gives the conclusion.

2. Related works

There are several works that use component based high abstraction level methodologies for defining embedded systems. MoPCoM [A. 08c] is a project that targets modeling and code generation of embedded systems using the block diagrams present in SysML which can be viewed as components. In [F. 08], a SynDEx based design flow is presented to manage SoC reconfigurability via implementation in FPGAs, with the application and architecture parts modeled as components. Similarly in [Gra08], a component based UML profile is described along with a tool set for modeling, verification and simulation of real-time embedded systems. Reconfiguration in SoC can be related to available system resources such as available memory, computation capacity and power consumption. An example of a component based approach with adaptation mechanisms is provided in [SA00]; e.g. for switching between resources [BAP05].

In [Lat99],[SKM01], the authors concentrate on verification of real-time embedded systems in which the control is specified at a high abstraction level via UML state machines and collaborations; by using model checking. However, control methodologies vary in nature as they can be expressed via different forms such as Petri Nets [B. 04], or other formalisms such as mode automata [MR98].

Mode automata extend synchronous dataflow languages with an imperative style, but without many modifications of language style and structure [MR98]. They are mainly composed of *modes* and *transitions*. In an automaton, each mode has the same interface. Equations are specified in modes. Transitions are associated with conditions, which serve to act as triggers. Mode automata can be composed together in either in parallel or hierarchical manner. They enable formal validation by using the synchronous technology. Among existing UML based approaches allowing for design verification are the Omega project [Gra08] and Diplodocus [AMAB$^+$06]. These approaches essentially utilize model checking and theorem proving.

In the domain of dynamically reconfigurable FPGA based SoCs, Xilinx initially proposed two design flows, which were not very effective leading to new alternatives. An effective modular approach for 2-D shaped reconfigurable modules was presented in [P. 05]. [J. 03] implemented modular reconfiguration using a horizontal slice based bus macro in order to connect the static and partial regions. They then placed arbitrary 2-dimensional rectangular shaped modules using routing primitives [M. 06]. This approach has been further refined in [C. 08]. In 2006, Xilinx introduced the *Early Access Partial Reconfiguration Design Flow* [Xil06] that integrated concepts of [P. 05] and [J. 03]. Works such as [Bay08],[K. 07] focus on implementing softcore internal configuration ports on Xilinx FPGAs such as Spartan-3, that do not have the hardware Internal Configuration Access Port (ICAP) reconfigurable core, for implementing PDR. Contributions such as introduced in [C. 07] and [A. 08a], illustrate usage of customized ICAPs. Finally in [R. 06], the ICAP reconfigurable core is connected with Networks-on-chip (NoC) implemented on dynamically reconfigurable FPGAs.

In comparison to the above related works, our proposition takes into account the following domains: SoC co-design, control/data flow, MDE, UML MARTE profile, SoC reconfigurability and PDR for FPGAs; which is the novelty of our design framework.

3. Components

Components are widely used in the domain of component based software development or component based software engineering. The key concept is to visualize the system as a collection of *components* [E. 03]. A widely accepted definition of components in software domain is given by Szyperski in [Szy98]:

A component is a unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently, and is subject to third-party composition.

In the software engineering discipline, a component is viewed as a representation of a self-contained part or subsystem; and serves as a building block for designing a complex global system. A component can provide or require *services* to its environment via well-specified *interfaces* [E. 03]. These interfaces can be related to *ports* of the component. Development of these components must be separated from the development of the system containing these modules. Thus components can be used in different contexts, facilitating their reuse.

The definition given by Szyperski permits to separate the component *behavior* and the component *interface*. Component *behavior* defines the functionality or the executable realization of a component. This can be viewed as associating the component with an *implementation* such as compilable code, binary form, etc.; depending upon the component model. This notion enables to link the component to user defined or third party implementations or *intellectual properties* (IPs). A component *interface* represents the properties of the component that are externally visible to other parts of the system.

Two basic prerequisites permit integration and execution of components. A *component model* defines the semantics that components must follow for their proper evolution [E. 03]. A *component infrastructure* is the design-time and run-time infrastructure that allows interaction between components and manages their assembly and resources. Obviously, there is a correspondence between a component model and the supporting mechanisms and services of a component framework.

Typically, in languages such as *Architecture Definition Languages* (ADLs), description of system architectures is carried out via compositions of hardware and software modules. These components follow a component model; and the interaction between components is managed by a component infrastructure [L. 98].

For describing hardware components in embedded systems, several critical properties, such as timing, performance and energy consumption, depend on characteristics of the underlying hardware platform. These extra functional properties such as performance cannot be specified for a *software* component but are critical for defining a hardware platform.

Component models

A component model determines the behavior of components within a component framework. It states what it means for a component to implement a given interface, it also imposes constraints such as defining communication protocols between components etc. [E. 03]. We have already briefly described the use of components in software engineering. There exit many component models such as COM (Component Object Model), CORBA, EJB and .NET. Each of these models have distinct semantics which may render them incompatible with other component models. As these models prove more and more useful for the design, development and verification of complex software systems, more and more research is being carried out by hardware designers in order to utilize the existing concepts present in software engineering for facilitating the development of complex hardware platforms.

Already hardware and system description languages such as VHDL and SystemC which support incremental modular structural concepts can be used to model embedded systems and SoCs in a modular way.

Component Infrastructure

A component infrastructure provides a wide variety of services to enforce and support component models. Using an simple analogy, components are to infrastructures what processes are to an operating system. A component infrastructure manages the resources shared by the different components [E. 03]. It also provides the underlying mechanisms that allow component interactions and final assembly. Components can be either *homogeneous*: having the same functionality model but not the same behavior; or *heterogeneous*. Examples of homogeneous components can be found in systems such as grids and cubes of computing units. In systems such at TILE64 [ea08], homogeneous instances of processing units are connected together by communication media. These types of systems are partially homogeneous concerning the computation units but heterogeneous in terms of their interconnections. Nowadays, modern embedded systems are mainly composed of heterogeneous components. Correct assembly of these components must be ensured to obtain the desired interactions. A lot of research has been carried out to ensure the correctness of interface composition in heterogeneous component models. Enriching the interface properties of a same component enables in addressing different aspects, such as timing and power consumption [DHJP08]. The semantics related to component assembly can be selected by designers according to their system requirements. The assembly can be either static or dynamic in nature.

Towards SoC co-design

It is obvious that in the context of embedded systems, information related to hardware platforms must be added to component infrastructures. Properties such as timing constraints and resource utilization are some of the integral aspects. However, as different design platforms use different component models for describing their customized components, there is a lack of consensus on the development of components for real-time embedded systems. Similarly interaction and interfacing of the components is another key concept.

Dynamic reconfiguration. Dynamic reconfiguration of component structure depends on the context required by designer and can be determined by different Quality-of-Service (QoS) criteria. The dynamic

aspects may require the integration of a *controller* component for managing the overall reconfiguration. The semantics related to component infrastructure must take into consideration several key issues: instantiation and termination of these components, deletion in case of user requirement etc. Similarly communication mechanisms such as message passing, operation calls can be chosen for inter and intra communication (in case of composition hierarchy) of components.

In case of embedded systems, a suitable example can be of FPGAs. These reconfigurable architectures are mainly composed of heterogeneous components, such as processors, memories, peripherals, I/O devices, clocks and communication media such as buses and Network-on-Chips. For carrying out internal dynamic reconfiguration, a controller component: in the form of a hard/soft core processor, can be integrated into the system for managing the overall reconfiguration process.

4. Gaspard2: a SoC Co-Design Framework

Gaspard2 [DaR09],[A. 08b] is a SoC co-design framework dedicated to parallel hardware and software and is based on the classical Y-chart [D.D83]. One of the most important features of Gaspard2 is its ability for system co-modeling at a high abstraction level. Gaspard2 uses the Model-Driven Engineering methodology to model real-time embedded systems using the UML MARTE profile [OMG]; and UML graphical tools and technologies such as Papyrus and Eclipse Modeling Framework.

Figure 1.1 shows a global view of the Gaspard2 framework. Gaspard2 enables to model *software applications*, *hardware architectures* and their *allocations* in a concurrent manner. Once models of software applications and hardware architectures are defined, the functional parts (such as application tasks and data) can be mapped onto hardware resources (such as processors and memories) via *allocation(s)*. Gaspard2 also introduces a *deployment* level that allows to link hardware and software components with intellectual properties (IPs). This level is elaborated later in section 7.

For the purpose of automatic code generation from high level models, Gaspard2 adopts MDE model transformations (*model to model* and *model to text* transformations) towards different execution platforms, such as targeted towards synchronous domain for validation and analysis purposes [GRY+08b]; or FPGA synthesis related to partial dynamic reconfiguration [QMMD09], as shown in Figure 1.1. Model transformation chains allow moving from high abstraction levels to low enriched levels. Usually, the initial high level models contain only domain-specific

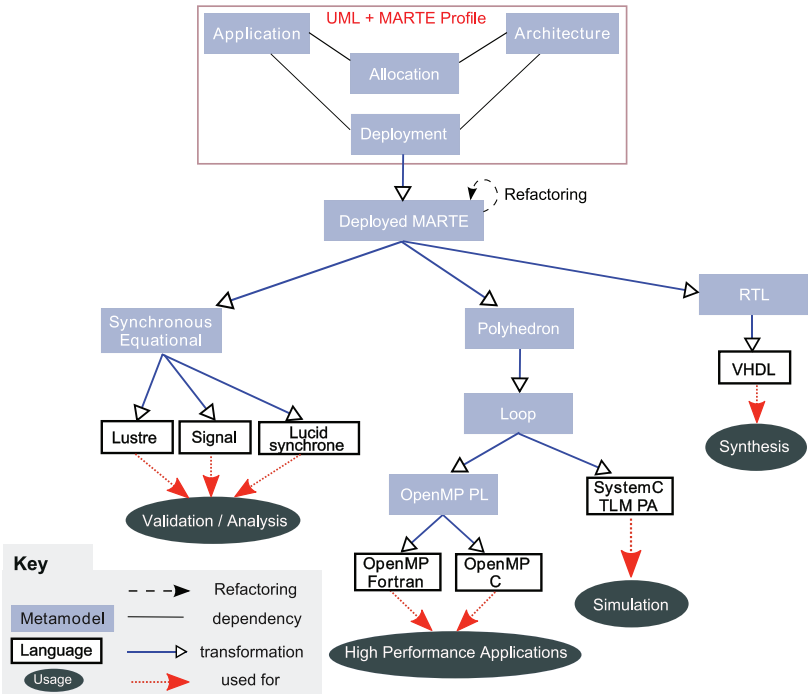


Figure 1.1. A global view of the Gaspard2 framework

concepts, while technological concepts are introduced seamlessly in the intermediate levels.

5. A reactive control model

We first describe the generic control semantics which can be integrated into the different levels (application, architecture and allocation) in SoC co-design. Several basic control concepts, such as `Mode Switch Component` and `State Graphs` are presented first. Then a basic composition of these concepts, which builds the mode automata, is discussed. This modeling derives from mode concepts in mode automata. The notion of exclusion among modes helps to separate different computations. As a result, programs are well structured and fault risk is reduced. We then use the Gaspard2 SoC co-design framework for utilization of these concepts.

Modes

A mode is a distinct method of operation that produces different results depending upon the user inputs. A mode switch component in Gaspard2 contains at least more than one mode; and offers a switch functionality that chooses execution of one mode, among several alternative present modes [O. 05]. The mode switch component in Figure 1.2 illustrates such a component having a *window* with multiple tabs and interfaces. For instance, it has an m (mode value input) port as well as several data input and output ports, i.e., i_d and o_d respectively. The switch between the different modes is carried out according to the mode value received through m.

The modes, M_1, ..., M_n, in the mode switch component are identified by the mode values: m_1, ..., m_n. Each mode can be hierarchical, repetitive or elementary in nature; and transforms the input data i_d into the output data o_d. All modes have the same interface (i.e. i_d and o_d ports). The activation of a mode relies on the reception of mode value m_k by the mode switch component through m. For any received mode value m_k, the mode runs exclusively. It should be noted that only mode value ports, i.e., m; are compulsory for creation of a mode switch component, as shown in Figure 1.2. Thus other type of ports are represented with dashed lines.

State graphs

A state graph in Gaspard2 is similar to state charts [Har87], which are used to model the system behavior using a state-based approach. It can be expressed as a graphical representation of transition functions as discussed in [GRY08a]. A state graph is composed of a set of vertices, which are called *states*. A state connects with other states through directed edges. These edges are called *transitions*. Transitions can be conditioned by some events or Boolean expressions. A special label *all*, on a transition outgoing from state s, indicates any other events that do not satisfy the conditions on other outgoing transitions from s. Each state is associated with some mode value specifications that provide mode values for the state. A state graph in Gaspard2 is associated with a `Gaspard State Graph` as shown in Figure 1.2.

Combining modes and state graphs

Once mode switch components and state graphs are introduced, a `MACRO` component can be used to compose them together. The MACRO in Figure 1.2 illustrates one possible composition. In this component,

the Gaspard state graph produces a mode value (or a set of mode values) and sends it (them) to the mode switch component. The latter switches the modes accordingly. Some data dependencies (or connections) between these components are not always necessary, for example, the data dependency between I_d and i_d. They are drawn with dashed lines in Figure 1.2. The illustrated figure is used as a basic composition, however, other compositions are also possible, for instance, one Gaspard state graph can control several mode switch components [QMD09].

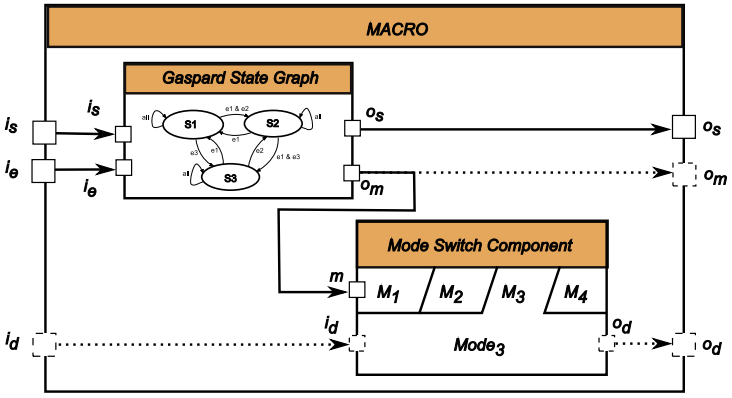


Figure 1.2. An example of a macro structure

6. Control at different system design levels

The previously mentioned control mechanisms can be integrated in different levels in a SoC co-design environment. We first analyze the control integration at the application, architecture and allocation levels in the particular case of the Gaspard2 framework, followed by a comparison of the three approaches.

Generic modeling concepts

We first present some concepts which are used in the modeling of mode automata. Gaspard2 uses the Repetitive Structure Modeling (RSM) package in the MARTE UML profile to model intensive data-parallel processing applications. RSM is based on Array-OL [Bou07] that describes the *potential parallelism* in a system; and is dedicated to data intensive multidimensional signal processing. In Gaspard2, data are manipulated in the form of multidimensional arrays. For an application functionality, both data parallelism and task parallelism can be expressed easily via RSM. A `repetitive component` expresses the data parallelism in an

application: in the form of sets of input and output `patterns` consumed and produced by the repetitions of the interior `part`. It represents a regular scalable component infrastructure. A `hierarchical` component contains several `parts`. It allows to define complex functionalities in a modular way and provides a structural aspect of the application. Specifically, task parallelism can be described using a hierarchical component in our framework.

The basic concepts of Gaspard2 control have been presented in Section V, but its complete semantics have not been provided. Hence, we propose to integrate mode automata semantics in the control. This choice is made to remove design ambiguity, enable desired properties and to enhance correctness and verifiability in the design. In addition to previously mentioned control concepts, three additional constructs as present in the RSM package in MARTE, namely the `Interrepetition` dependency (IRD), the `tiler` connector and `defaultLink` are used to build mode automata.

A `tiler` connector describes the tiling of produced and consumed arrays and thus defines the shape of a data pattern. The `Interrepetition` dependency is used to specify an acyclic dependency among the repetitions of the same component, compared to a tiler, which describes the dependency between the repeated component and its owner component. The interrepetition dependency specification leads to the sequential execution of repetitions. A `defaultLink` provides a default value for repetitions linked with an interrepetition dependency, with the condition that the source of dependency is absent.

The introduction of an interrepetition dependency serializes the repetitions and data can be conveyed between these repetitions. Hence, it is possible to establish mode automata from Gaspard2 control model, which requires two subsequent steps. First, the internal structure of `Gaspard Mode Automata` is presented by the `MACRO` component illustrated in Figure 1.2. The Gaspard state graph in the macro acts as a state-based controller and the mode switch component achieves the mode switch function. Secondly, interrepetition dependency specifications should be specified for the macro when it is placed in a repetitive context. The reasons are as follows. The macro structure represents only a single transition between states. In order to execute continuous transitions as present in automata, the macro should be repeated to have multiple transitions. An interrepetition dependency forces the continuous sequential execution. This allows the construction of mode automata which can be then executed.

Application level

With previous presented constructs, the modeling of Gaspard mode automata, which can be eventually translated into synchronous mode automata [MR98], is illustrated with an example in Figure 1.3, where the assembly of these constructs is presented. An interrepetition dependency connects the repetitions of `MACRO` and conveys the current state. It thus sends the target state of one repetition as the source state for the next repetition of the macro component as indicated by the value of -1. The states and transitions of the automata are encapsulated in the Gaspard state graph. The data computations inside a mode are set in the mode switch component. The detailed formal semantics related to Gaspard mode automata can be found in [GRY08a]. It should be noted that parallel and hierarchical mode automata can also be constructed using the control semantics.

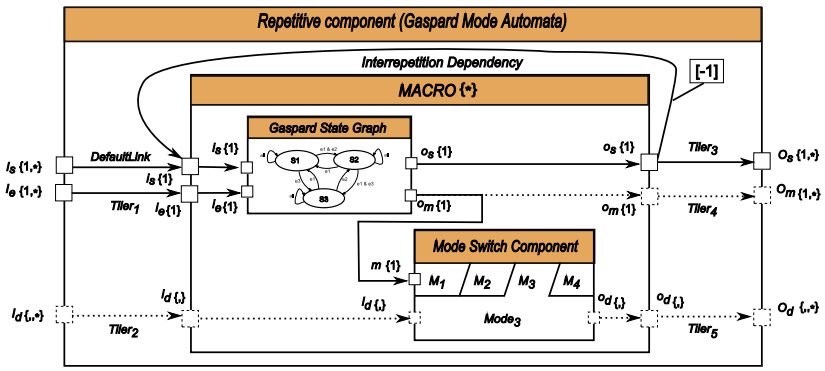


Figure 1.3. The macro structure in a repetitive component

The proposed control model enables the specification of system reconfigurability at the application level [H. 08]. Each mode in the switch can have different effects with regards to environmental or platform requirements. A mode represents a distinct algorithm to implement the same functionality as others. Each mode can have a different demand of memory, CPU load, etc. Environmental changes/platform requirements are captured as events; and taken as inputs of the control.

Architecture level

Gaspard2 uses the *Hardware Resource Modeling* (or HRM) package in the MARTE profile in combination with the RSM package to model large regular hardware architectures (such as multiprocessor architectures) in

a compact manner. Complex interconnection topologies can also be modeled via Gaspard2 [I.-08].

Control semantics can also be applied on to the architectural level in Gaspard2. As compared to the integration of control in other modeling levels (such as application and allocation), the control in architecture is more flexible and can be implemented in several forms. A controller can modify the structure of the architecture in question such as modifying the communication interconnections. The structure can be either modified globally or partially. In case of a global modification, the reconfiguration is viewed as static and the controller is present exterior to the targeted architecture. If the controller is present inside the architecture, then the reconfiguration is partial and could result in partial dynamic reconfiguration. However, the controller can be related to both the structural and behavioral aspects of the architecture. An example can be of a controller unit present inside a processing unit in the architecture for managing *Dynamic frequency scaling* [YHBM02] or *Dynamic voltage scaling* [IKH01]. These techniques allow power conservation by reducing the frequency or the voltage of an executing processor.

Allocation level

Gaspard2 uses the Allocation Modeling package (Alloc) to allocate SoC applications on to the targeted hardware architectures. Allocation in MARTE can be either *spatial* or *temporal* in nature [OMG].

Control at the allocation level can be used to decrease the number of active executing computing units to reduce the overall power consumption levels. Tasks of an application that are executing parallely on processing units may produce the desired computation at an optimal processing speed, but might consume more power, depending upon the inter-communication between the system. Modification of the allocation of the application on to the architecture can produce different combinations and different end results. A task may be switched to another processing unit that consumes less power, similarly, all tasks can be associated on to a single processing unit resulting in a temporal allocation as compared to a spatial one. This strategy may reduce the power consumption levels along with decrease in the processing speed. Thus allocation level allows to incorporate *Design Space Exploration* (DSE) aspects which in turn can be manipulated by the designers depending upon their chosen QoS criteria.

Comparison of control at the three levels

Integrating control at different aspects of system (application, architecture and allocation) has its advantages and disadvantages as briefly shown in the Figure 1.4. With respect to control integration, we are mainly concerned with several aspects such as the range of impact on other modeling levels. We define the impact range as either *local* or *global*, with the former only affecting the concerned modeling level while the later having consequences on other modeling levels. These consequences may vary and cause changes in either *functional* or *non-functional aspects* of the system. The modification in application may arise due to QoS criteria such as switching from a high resolution mode to a lower one in a video processing functionality. However, the control model may have consequences, as change in an application functionality or its structure may not have the intended end results.

Control integration in an architecture can have several possibilities. The control can be mainly concerned with modification of the hardware parameters such as voltage and frequency for manipulating power consumption levels. This type of control is local and mainly used for QoS, while the second type of control can be used to modify the system structure either globally or partially. This in turn can influence other modeling levels such as the allocation. Thus allocation needs to be modified every single time when there is a modification in the structure of the execution platform.

	Impact on other models	Conditions	Consequences
Application	Local	—	Change in application functionality/structure occurs, the functionality can be related to QoS criteria as well
Architecture	Local	QoS variation only	Variability in observed performance
	Global	Modification in structure	Allocation model needs to be modified as well
Allocation	Local	Other models are fixed	Variability in observed performance
	Global	Other models modifiable	Change in functionality, variation in QoS possible

Figure 1.4. Overview of control on the first three levels of a SoC framework

Control at the allocation is local only when both the application and architecture models have been pre-defined to be static in nature which is rarely the actual scenario. If either the application or the architecture is changed, the allocation must be adapted accordingly.

It is also possible to form a merged control by combining the control models at different aspects of the system to form a mixed-level control approach. However, detailed analysis is needed to ensure that any combination of control levels does not cause any unwanted consequences. This is also a tedious task. During analysis, several aspects have to be

monitored, such as ensuring that no conflicts arise due to a merged approach. Similarly, redundancy should be avoided: if an application control and architecture control produce the same result separately; then suppression of control from one of these levels is warranted. However, this may also lead to an instability in the system. It may be also possible to create a global controller that is responsible for synchronizing various local control mechanisms. However, clear semantics must be defined for the composition of the global controller which could lead to an overall increase in design complexity.

The global impact of any control model is undesirable as the modeling approach becomes more complex and several high abstraction levels need to be managed. A local approach is more desirable as it does not affect any other modeling level. However, in each of the above mentioned control models, strict conditions must be fulfilled for their construction. These conditions may not be met depending upon the designer environment. Thus an ideal control model is one that has only a local impact range and does not have any strict construction conditions.

7. Control at Deployment level

In this section we explain control integration at another abstraction level in SoC co-design. This level deals with linking the modeled application and architecture components to their respective IPs. We explain the component model of this *deployment* level in the particular case of the Gaspard2 framework within the context of dynamic reconfiguration.

For dynamic reconfiguration in modern SoCs, an embedded controller is essential for managing a dynamically reconfigurable region. This component is usually associated with some control semantics such as state machines, Petri nets etc. The controller normally has two functionalities: one responsible for communicating with the FPGA *Internal Configuration Access Port* hardware reconfigurable core or ICAP [B. 03] that handles the actual FPGA switching; and a state machine part for switching between the available configurations. The first functionality is written manually due to some low level technological details which cannot be expressed via a high level modeling approach.

The control at the deployment level is utilized to generate the second functionality automatically via model transformations. Finally the two parts can be used to implement partial dynamic reconfiguration in an FPGA that can be divided into several static/reconfigurable regions. A reconfigurable region can have several implementations, with each having the same interface, and can be viewed as a mode switch component with different modes. In our design flow, this dynamic region

is generated from the high abstraction levels, i.e., a complex Gaspard2 application specified using the MARTE profile. Using the control aspects in the subsequently explained Gaspard2 deployment level, it is possible to create different configurations of the modeled application. Afterwards, using model transformations, the application can be transformed into a hardware functionality, i.e., a dynamically reconfigurable hardware accelerator, with the modeled application configurations serving as different implementations related to the hardware accelerator.

We now present integration of the control model at the deployment level. We first explain the deployment level in Gaspard and our extensions followed by the control model.

Deployment in Gaspard2

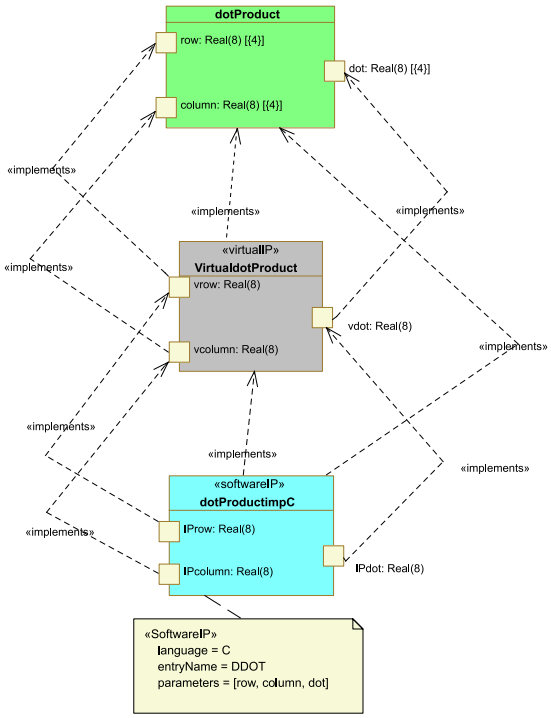


Figure 1.5. Deployment of an elementary `dotProduct` component in Gaspard2

The Gaspard2 deployment level enables one to precise a specific IP for each elementary component of application or architecture, among several possibilities [APN+07]. The reason is that in SoC design, a functionality can be implemented in different ways. For example, an application functionality can either be optimized for a processor, thus

written in C/C++, or implemented as a hardware accelerator using *Hardware Description Languages* (HDLs). Hence the deployment level differentiates between the hardware and software functionalities; and allows moving from platform-independent high level models to platform-dependent models for eventual implementation. We now present a brief overview of the deployment concepts.

A `VirtualIP` expresses the functionality of an elementary component, independently from the compilation target. For an elementary component K, it associates K with all its possible IPs. The desired IP(s) is (are) then selected by the SoC designer by linking it (them) to K via an `implements` dependency. Finally, the `CodeFile` (not illustrated in the chapter) determines the physical path related to the source/binary code of an IP, along with required compilation options.

Multi-Configuration approach

Currently in deployment level, an elementary component can be associated with only one IP among the different available choices (if any). Thus the result of the application/architecture (or the mapping of the two forming the overall system) is a static one. This collective composition is termed as a *Configuration*.

Integrating control in deployment allows to create several configurations related to the modeled application for the final realization in an FPGA. Each configuration is viewed as a collection of different IPs, with each IP associated with its respective elementary component. The end result being that one application model is transformed by means of model transformations and intermediate metamodels into a dynamically reconfigurable hardware accelerator, having different implementations equivalent to the modeled application configurations.

A `Configuration` has the following attributes. The name attribute helps to clarify the configuration name given by a SoC designer. The ConfigurationID attribute permits to assign unique values to each of the modeled `Configuration`, which in turn are used by the control aspects presented earlier. Theses values are used by a Gaspard state graph to produce the mode values associated with its corresponding Gaspard state graph component. These mode values are then sent to a mode switch component which matches the values with the names of its related collaborations as explained in [QMD09]. If there is a match, the mode switch component switches to the required configuration. The InitialConfiguration attribute sets a Boolean value to a configuration to indicate if it is the initial configuration to be loaded onto the target

FPGA. This attribute also helps to determine the initial state of the Gaspard state graph.

An elementary component can also be associated with the same IP in different configurations. This point is very relevant to the semantics of partial bitstreams, e.g., FPGA configuration files for partial dynamic reconfiguration, supporting *glitchless dynamic reconfiguration*: if a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation. If the same IP for an elementary component is present in several configurations, that IP is not changed during reconfiguration. It is thus possible to link several IPs with a corresponding elementary component; and each link relates to a unique configuration. We apply a condition that for any n number of configurations with each having m elementary components, each elementary component of a configuration must have *at least* one IP. This allows successful creation of a complete configuration for eventual final FPGA synthesis.

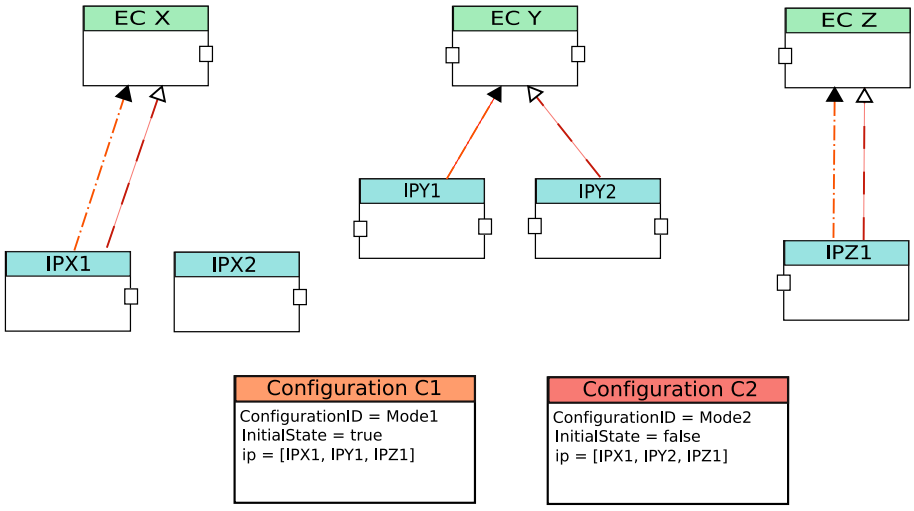


Figure 1.6. Abstract overview of configurations in deployment

Figure 1.6 represents an abstract overview of the configuration mechanism introduced at the deployment level. We consider a hypothetical Gaspard2 application having three elementary components *EC X*, *EC Y* and *EC Z*, having available implementations *IPX1*, *IPX2*, *IPY1*, *IPY2* and *IPZ1* respectively. For the sake of clarity, this abstract representation omits several modeling concepts such as `VirtualIP` and `Implements`. However, this representation is very close to UML model-

ing as presented earlier in the chapter. A change in associated implementation of any of these elementary components may produces a different end result related to the overall functionality, and different QoS criteria such as effectively consumed FPGA resources.

Here two configurations *Configuration C1* and *Configuration C2* are illustrated in the figure. *Configuration C1* is selected as the initial configuration and has associated IPs: *IPX1*, *IPY1* and *IPZ1*. Similarly *Configuration C2* also has its associated IPs. This figure illustrates all the possibilities: an IP can be globally or partially shared between different configurations: such as *IPX1*; or may not be included at all in a configuration, e.g., case of *IPX2*.

Once the different implementations are created by means of model transformations, each implementation is treated as a source for a partial bitstream. A bitstream contains packets of FPGA configuration control information as well as the configuration data. Each partial bitstream signifies a unique implementation, related to the reconfigurable hardware accelerator which is connected to an embedded controller. While this extension allows to create different configurations, the state machine part of the controller is created manually. For automatic generation of this functionality, the deployment extensions are not sufficient. We then make use of the earlier control semantics at the deployment level.

Implementation

Once control has been integrated at deployment level, it helps to switch between the different modeled configurations [QMMD09]. The configurations relate to a Gaspard2 application modeled at the high abstraction levels. This application is transformed into a hardware functionality, i.e., a hardware accelerator, by means of the model transformations, as stated earlier.

The application targeted for the validation of our methodology is a delay estimation correlation module integrated in an anti-collision radar detection system. Our radar uses a PRBS (Pseudorandom binary sequence) of length of 127 chips. In order to produce a computation result, the algorithm requires 127 multiplications between the 127 elements of the reference code that is generated via MATLAB and the last 127 received samples. The result of this multiplication produces 64 data elements. The sum of these 64 data elements produces the final result. This result can be sent as input to other parts of our radar detection system [QEMD09] in order to detect the nearest object. The different configurations related to our application change the IPs related to the elementary components, which in turn allow us to manipulate

different QoS criteria such as consumed FPGA resources and overall energy consumption levels. The partially reconfigurable system has been implemented on a Xilinx XC2VP30 Virtex-II Pro FPGA with a hardcore PowerPC 405 processor as a reconfiguration controller with a frequency of 100 MHz. We implemented two configurations on the targeted architecture, two with different IPs related to an multiplication elementary component in the application and a blank configuration. The results are shown in Figure 1.7.

	FPGA resources			Dynamic Power (mW)	Mean Reconfiguration Time (secs)
	Slices	Slice Flip Flops	LUTs		
Configuration 1 : Multiplication IP written with DSP functionality	1272/13696 (9.287%)	2084/27392 (7.608%)	1584/27392 (5.782%)	221.42	1.45
Configuration 2 : Multiplication IP written with if-else functionality	1186/13696 (8.659%)	1944/27392 (7.096%)	1836/27392 (6.702%)	214.56	1.41

Figure 1.7. An overview of the obtained results

Advantages of control deployment level

The advantage of using control at deployment is that the impact level remains local and there is no influence on other modeling levels. Another advantage is that the application, architecture and allocation models can be reused again and only the necessary IPs are modified. As we validate our methodology by implementing partial dynamic reconfigurable FPGAs, we need to clarify about the option of choosing mode-automata.

Many different approaches exist for expressing control semantics, mode automata were selected as they clearly separate control/data flow. They also adapt a state based approach facilitating seamless integration in our framework; and can be expressed at the MARTE specification levels. The same control semantics are then used throughout our framework to provide a single homogeneous approach. With regards to partial dynamic reconfiguration, different implementations of a reconfigurable region must have the same external interface for integration with the static region at run-time. Mode automata control semantics can express the different implementations collectively via the concept of a mode switch, which can be expressed graphically at high abstraction levels using the concept of a mode switch component. Similarly a state graph component expresses the controller responsible for the context switch between the different configurations.

8. Conclusion

This chapter presents a high abstraction level component based approach integrated in Gaspard2, a SoC co-design framework compliant with the MARTE standard. The control model is based on mode automata, and takes task and data parallelism into account. The control semantics can be integrated into various levels in Gaspard2. We compare the different approaches with respect to different criteria such as impact on other modeling levels. Control integration in application level allows dynamic context switching. In addition, safety of the control can be checked by tools associated with synchronous languages when the high-level model is transformed into synchronous code. Control at the architectural level can be concerned with QoS criteria as well as structural aspects. Similarly, control at the allocation level offers advantages of *Design Space Exploration*. Finally we present control semantics in the deployment level which offer reuse of application, architecture and allocation models. This control model makes it possible to support partial dynamic reconfiguration in reconfigurable FPGAs. A case study has also been briefly presented to validate our design methodology. Currently we have only focused on isolating controls at different levels in Gaspard2. An ideal perspective could be a combination of the different control models to form a merged approach.

References

[A. 08a] A. Cuoccio and P. R. Grassi and V. Rana and M. D. Santambrogio and D. Sciuto. A Generation Flow for Self-Reconfiguration Controllers Customization. *Forth IEEE International Symposium on Electronic Design, Test and Applications, DELTA 2008*, pages 279–284, 2008.

[A. 08b] A. Gamatié and S. Le Beux and E. Piel and A. Etien and R. B. Atitallah and P. Marquet and J.-L Dekeyser. A model driven design framework for high performance embedded systems. Research Report RR-6614, INRIA, 2008. `http://hal.inria.fr/inria-00311115/en`.

[A. 08c] A. Koudri et al. Using MARTE in the MOPCOM SoC/SoPC Co-Methodology. In *MARTE Workshop at DATE'08*, 2008.

[AMAB+06] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A UML-based environment for system design space exploration. *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pages 1272–1275, Dec. 2006.

[APN+07] R. B. Atitallah, E. Piel, S. Niar, P. Marquet, and J.-L Dekeyser. Multilevel MPSoC simulation using an MDE approach. In *SoCC 2007*, 2007.

[B. 03] B. Blodget and S. McMillan and P. Lysaght. A lightweight approach for embedded reconfiguration of FPGAs. In *Design, Automation & Test in Europe, DATE'03*, 2003.

[B. 04] B. Nascimento et al. A partial reconfigurable architecture for controllers based on Petri nets. In *SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design*, pages 16–21. ACM, 2004.

[BAP05] Jérémy Buisson, Franoise André, and Jean-Louis Pazat. A Framework for Dynamic Adaptation of Parallel Components. In *ParCo 2005*, 2005.

[Bay08] Bayar, S., and Yurdakul, A. Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP). In *2nd HiPEAC workshop on Reconfigurable Computing, HiPEAC 08*, 2008.

[Bou07] P. Boulet. Array-OL revisited, multidimensional intensive signal processing specification. Research Report RR-6113, INRIA, `http://hal.inria.fr/inria-00128840/en/`, February 2007.

[C. 07] C. Claus and F.H. Muller and J. Zeppenfeld and W. Stechele. A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration. *IPDPS 2007*, pages 1–7, 2007.

[C. 08] C. Schuck and M. Kuhnle and M. Hubner and J. Becker. A framework for dynamic 2D placement on FPGAs. In *IPDPS 2008*, 2008.

[DaR09] DaRT team. GASPARD SoC Framework, 2009. `http://www.gaspard2.org/`.

[D.D83] D.D. Gajski and R. Khun. New VLSI Tools. *IEEE Computer*, 16:11–14, 1983.

[DHJP08] L. Doyen, T. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *EMSOFT'08: Proceedings of the 8th ACM international conference on Embedded software*, pages 79–88, New York, NY, USA, 2008. ACM.

[E. 03] E. Brinksma and G. Coulson and I. Crnkovic and A. Evans and S. Grard and S. Graf and H. Hermanns and J. Jzquel and B. Jonsson and A. Ravn and P. Schnoebelen and F. Terrier and A. Votintseva. Component-based Design and Integration Platforms: a Roadmap. *The Artist consortium*, 2003.

[ea08] S. Bell et al. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *IEEE International Digest of Technical Pa-*

pers on Solid-State Circuits Conference (ISSCC 2008), pages 88–598, 2008.

[F. 08] F. Berthelot and F. Nouvel and D. Houzet. A Flexible system level design methodology targeting run-time reconfigurable FPGAs. *EURASIP Journal of Embedded Systems*, 8(3):1–18, 2008.

[Gra08] S. Graf. Omega – Correct Development of Real Time Embedded Systems. *SoSyM, int. Journal on Software & Systems Modelling*, 7(2):127–130, 2008.

[GRY08a] A. Gamatié, É. Rutten, and H. Yu. A Model for the Mixed-Design of Data-Intensive and Control-Oriented Embedded Systems. Research Report RR-6589, INRIA, `http://hal.inria.fr/inria-00293909/fr`, July 2008.

[GRY+08b] A. Gamatié, É. Rutten, H. Yu, P. Boulet, and J.-L. Dekeyser. Synchronous modeling and analysis of data intensive applications. *EURASIP Journal on Embedded Systems*, 2008. To appear. Also available as INRIA Research Report: `http://hal.inria.fr/inria-00001216/en/`.

[H. 08] H. Yu. *A MARTE based reactive model for data-parallel intensive processing: Transformation toward the synchronous model*. PhD thesis, USTL, 2008.

[Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987. article note found.

[I.-08] I.-R. Quadri and P. Boulet and S. Meftali and J.-L. Dekeyser. Using An MDE Approach for Modeling of Interconnection networks. In *The International Symposium on Parallel Architectures, Algorithms and Networks Conference (ISPAN 08)*, 2008.

[IKH01] C. Im, H. Kim, and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers, 2001.

[J. 03] J. Becker and M. Huebner and M. Ullmann. Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations. In *VLSI'03*, 2003.

[K. 07] K. Paulsson and M. Hubner and G. Auer and M. Dreschmann and L. Chen and J. Becker. Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGA. *International Conference on Field Programmable Logic and Applications, FPL 2007*, pages 351–356, 2007.

[L. 98] L. Bass and P. Clements and R. Kazman. Software Architecture In Practice. *Addison Wesley*, 1998.

[Lat99] Latella, D. and Majzik, I. and Massink, M. Automatic Verification of a Behavioral Subset of UML Statechart Diagrams Using the SPIN Model-Checker. In *Formal Aspects Computing*, volume 11, pages 637–664, 199.

[M. 06] M. Huebner and C. Schuck and M. Kiihnle and J. Becker. New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-Time Adaptive Microelectronic Circuits. In *ISVLSI'06*, 2006.

[MR98] F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. In *European Symposium On Programming*, Lisbon (Portugal), March 1998. Springer verlag.

[O. 05] O. Labbani and J.-L Dekeyser and Pierre Boulet and É. Rutten. Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach. In *Proceedings of the International Workshop MARTES: Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[OMG] OMG. Modeling and analysis of real-time and embedded systems (MARTE). http://www.omgmarte.org/.

[OMG07] OMG. Portal of the Model Driven Engineering Community, 2007. `http://www.planetmde.org`.

[P. 05] P. Sedcole and B. Blodget and J. Anderson and P. Lysaght and T. Becker. Modular Partial Reconfiguration in Virtex FPGAs. In *International Conference on Field Programmable Logic and Applications, FPL'05*, pages 211–216, 2005.

[P. 06] P. Lysaght and B. Blodget and J. Mason. Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs. In *FPL'06*, 2006.

[QEMD09] I. R. Quadri, Yassin Elhillali, S. Meftali, and J.-L. Dekeyser. Model based design flow for implementing an Anti-Collision Radar system. In *9th International IEEE Conference on ITS Telecommunications (ITS-T 2009)*, 2009.

[QMD09] I. R. Quadri, S. Meftali, and J.-L. Dekeyser. Integrating Mode Automata Control Models in SoC Co-Design for Dynamically Reconfigurable FPGAs. In *International Conference on Design and Architectures for Signal and Image Processing (DASIP 09)*, 2009.

[QMMD09] I. R. Quadri, A. Muller, S. Meftali, and J.-L. Dekeyser. MARTE based design flow for Partially Reconfigurable

Systems-on-Chips. In *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)*, 2009.

[R. 06] R. Koch and T. Pionteck and C. Albrecht and E. Maehle. An adaptive system-on-chip for network applications. In *IPDPS 2006*, 2006.

[SA00] M.T. Segarra and F. André. A framework for dynamic adaptation in wireless environments. In *Proceedings of 33rd International Conference on Technology of Object-Oriented Languages (TOOLS 33)*, pages 336–347, 2000.

[SKM01] T. Schäfer, A. Knapp, and S. Merz. Model checking UML state machines and collaborations. In *CAV Workshop on Software Model Checking*, ENTCS 55(3), 2001.

[Szy98] C. Szyperski. ACM Press and Addison-Wesley, New York, N.Y, 1998.

[T. 06] T. Mens and P. Van Gorp. A taxonomy of model transformation. In *Proceedings of the International Workshop on Graph and Model Transformation, GraMoT 2005*, pages 125–142, 2006.

[Xil06] Xilinx. Early Access Partial Reconfigurable Flow. 2006. `http://www.xilinx.com/support/prealounge/protected/index.htm`.

[YHBM02] L. Yung-Hsiang, L. Benini, and G. De Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1284–1305, 2002.

IP Based Design 2003

Session: SystemC for IP Modeling

A FAST SYSTEMC SIMULATION METHODOLOGY FOR MULTILEVEL IP/SOC DESIGN

Samy Meftali, Joel Vennin and Jean-Luc Dekeyser, LIFL
{meftali, vennin, dekeyser}@lifl.fr

Lille, France

Abstract:

In this paper, we present a fast method which allows connecting together SystemC modules. These modules may be specified at different abstraction levels, and we obtain an executable simulation model of the whole system. The originality of our work is the fact that it does not require SystemC external libraries. Indeed, the simulation model generation is based only on: modules specifications, a SystemC added library and a simple rules description language. Thus with our methodology, we generate simulation module adapters having a simple structure, in a fast and automatic way, by instantiating a generic configurable SystemC class.

1. INTRODUCTION

Generally before obtaining a SoC on silicium, systems are specified at several abstraction levels (functional, logical, RT,..etc), each specification is called a model. Any system design flow consists in refining, more or less automatically, each model to obtain another, starting from a functional model until Register Transfer level (RTL) model.

So each design tool must contain simulation engines in order to validate each model before refining it [Mef02]. But, as complex systems are mainly designed by assembling existing components (IPs), designers want sometimes to just add some functionality to a system constituted by a set of already existing IPs to obtain scalable SoSs. In the case that this initial system is available only at RT level for example, it can be very benefic, in terms of design time, to design only the part to add to the system from the functional level, and not to specify again all the application at high abstraction levels. However, a multilevel simulation platform will be needed to validate the whole system (the part to add at a high abstraction level, and the old application at a low level). This constitutes the objective of our work.

This paper is organized as follow: in section 2, we present some necessary basics to understand this work. The related works are presented in the section 3. Section 4 presents the different steps of our multi level simulation model generation. Our approach is applied to a system containing an SDRAM described at RT level and a processor module described at UTF level in the section 5. We conclude this paper in the section 6.

2. BASICS

The multi level simulation methodology that we present in this paper concerns exclusively systems described entirely in SystemC. We present in this section some basics necessary to understand how our approach works.

2.1. Abstraction levels

SystemC design flow allows description of system modules (IPs) mainly at four abstraction levels [Gro02]. The general characteristics of each abstraction level are described in the following paragraphs.

2.1.1. Untimed Functional Level (UTF)

At this level a model is similar to an executable specification, but no time delays at all are present in the model. Shared communication links (as buses) are not modeled at UTF level. The communication between modules is point-to-point, and usually modeled using FIFOs with blocking write and read methods.

2.1.2. Timed Functional Level (TF)

A TF model is similar to UTF one in that communication between modules is still point-to-point, and there is no shared communication links. However, at this abstraction level timing delays are added to processes within the design to reflect

timing constraints of the specification and also processing delays of target architecture.

2.1.3. Transaction Level

In a transaction level specification, communication between modules is modeled using function calls. At this level the communication model is accurate in terms of functionality and often in terms of timing. For example in a SoC transaction level specification, we may model the different types of transactions that the on-chip bus supports, as burst read/write transactions, but we don't model the pins of the modules that connect to the bus.

2.1.4. Register Transfer Level

It is the lowest abstraction level in SystemC systems design flow. The internal structure of an RT level model accurately reflects the registers and combinatorial logic of the target architecture. The communication between modules is described in details in terms of used protocols and timing. Each module's behavior corresponds exactly to a physical component behavior. The data types used at RT level are mainly bits (or bit-vectors).

2.2. Multi level simulation

A multi level simulation model is an executable specification containing a set of modules described at different abstraction levels. The problem in such a model is how to connect automatically and with low-cost two (or many) modules communicating each one using different data types via different communication protocols. Figure 1 shows an example of a system containing two modules: module 1 and module 2. The first one is specified at UTF level and communicates throw 3 ports using blocking FIFOs with integer data, the second one communicates throw 11 ports using several protocols (Master-Slave, Hand-shack,..etc) with bit and bit vector types. Thus the problem is how to convert data types and protocols to allow the communication between these two modules.

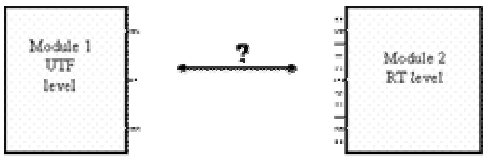


Figure 1: Multilevel simulation problem

3. RELATED WORK

Recently many academic and industrial research teams worked on the multi level simulation problem in embedded systems. This type of simulation implies to execute together models of the system component described at different abstraction levels. Among the proposed solutions, the Bus Functional Model (BFM) [Sem00] constitutes the conventional methodology to interconnect functional simulation models and cycle accurate models, especially to validate software/hardware interfaces. Unfortunately this methodology takes into account only memory accesses, but it doesn't at all allow transformation of high level communication primitives (FIFO for example).

CoWareN2C [Cow02] is an environment which offers a multi level cosimulation solution. It allows the use of two abstraction levels: BCA (Bus Cycle Accurate) which is closely similar to RTL and UT level (without timing references) Thus CoWareN2C presents a concept called BCASH, It is a wrapper of the sub-systems described at UT level allowing the estimation of their sub routines execution time. This wrapper can be automatically generated only in if the sub-system at UT level is targeted in software.(that means that the sub-system will be executed on a processor simulator ISS "Instruction Set Simulator"). This is unfortunately a very strong constraint.

The concept of conversion interface is present in SystemC [Sys]. It may permits the connection of modules communicating by Remote Procedure Calls (RPC) with other module described at RT level. This interface must be entirely hand coded by the designer and this may be of course very time consuming and error prone.

We can find some other multi-level simulation environments as Chinook [Cho95] focusing on the dynamicity while changing modules abstraction levels.

VSI Alliance [Vsi] works on heterogeneous components assembling. It presents a specification and documentation standard, at different abstraction levels and also a bus interface standard. But it doesn't focus yet on the simulation adapters' generation. Thus COSY [Bru00] is a generic model which allows the interconnection of IPs via VCI bus model interfaces, but it still focus on the system level.

Some very recent works, in the literature, treat the problem of automatic generation of multi level simulation models for heterogeneous multiprocessor SoC.

The work described in [Nic02] is one significative example. It permits to generate automatically simulation wrappers to adapt modules abstraction levels to the simulation level. Unfortunately, it doesn't target a specific class of application, and the wrappers are constructed by assembling basic components from external libraries. The structure of these simulation wrappers seems to be complex because of the important number of SystemC components (processes) present in each instance.

Our contribution is the proposal of a new methodology to validate SoCs by simulation. With

this approach we can perform a fast and low cost simulation on systems constructed by assembling IPs. The IPs can be described at different abstraction level. The main originality of our approach is that it does not need any external SystemC libraries. It uses only internal SystemC libraries and a small rules description language.

4. METHODOLOGY

Our simulation models generation flow generates a simulation module adapter for each module described at an abstraction level different from those on which we want to perform de simulation of the system. If we have for example a SoC composed by two modules: module 1 specified at UTF level and module 2 specified at RT level, and we want to simulate it at the RT level, our methodology generates automatically a simulation module adapter for the module 1 in order to adapt it's interface with those of module.

4.1. Simulation module adapter's structure

Our simulation module adapter is constituted mainly by three parts: real ports, logical ports and an infinite SystemC process as shown in the Figure 2. These parts are described with details in the remaining of this section.

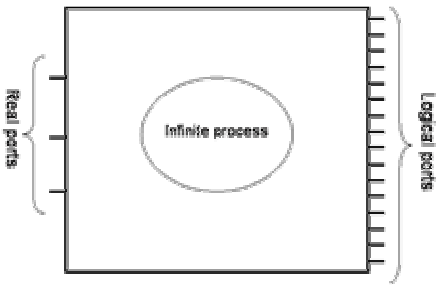


Figure 2: Simulation module adapter's structure

4.1.1. Real ports

They correspond to the initial module ports. They have exactly the same characteristics of the real module's ports (data types, communication protocols,..).

4.1.2. Logical ports

A simulation module adapter must have the same number of logical ports as the number of real ports of the module with which it will be connected. Each one of these logical ports must have identical characteristics as the real port of the module with which it will be connected.

4.1.3. Infinite process

It is an automatically generated SystemC process, by instantiating and configuring a generic SystemC class that we defined. It is sensitive to the signals connecting it to the real ports. This process reads data from the real ports, converts them to be adequate with the logical ports types.

The automatic generation methodology of the simulation module adapters is presented in the remaining of this section.

4.2. Multilevel system simulation model generation flow

Having the interface of each one of the modules to be connected, our flow produces a multi level simulation model as shown in the Figure 3.

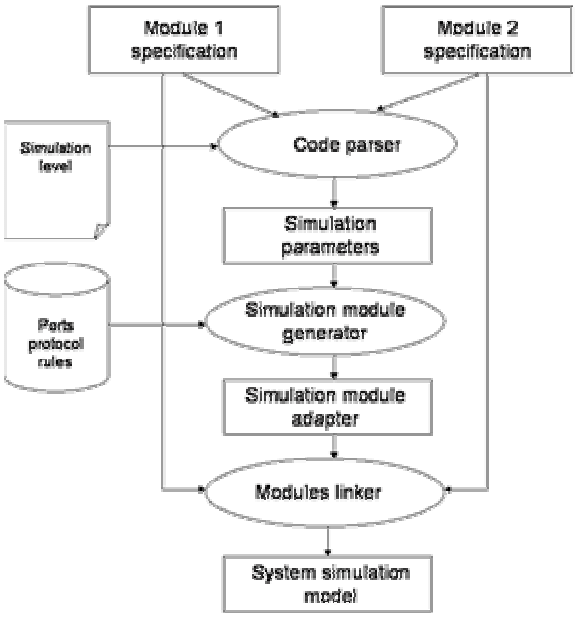


Figure 3: Multi level system simulation model generation flow

Our simulation module adapters are not constructed by assembling basic components for external libraries as in almost literature tools, but it is generated by rules composition. In our simulation models generation flow all simulation module adapters are instances of a SystemC class that we created and called "generic_interface".

The generic interface is a configurable SystemC class. It is composed, as all SystemC classes, by an interface file (.h) and an implementation/behavior file (.cc). The interface file defining the ports characteristics is automatically configured from the initial system specification, and the behavior file is constructed and configured by the composition of a set of rules. The rules composition is described in details in this section.

We distinguish mainly three engines in our simulation generation flow (Figure 3).

4.2.1. Code parser

The code parser engine takes the SystemC interface files (.h) of the modules to be connected, and the simulation level from the designer. It produces the simulation parameters. These

parameters are mainly the number of the logical and real ports, the communication protocol of each one of them and the simulation level at which the designer want perform the system simulation.

4.2.2. Simulation module generator

After obtaining all the simulation parameters (modules interfaces and simulation level), the simulation module generator engine instantiates and configures a generic class to obtain a SystemC module adapting a given module to the simulation level. All necessary simulation adapters are produced by instantiating GenericInterface. This makes the final simulation model's structure simple and its generation easy.

A module adapter is composed by an interface (describing the ports of the module adapter) and a behavior files (a SystemC infinite process describing the behavior of the simulation module adapter). The first one is produced using association rules and the second one using mainly transformation rules. These two kinds of rules are described in the next paragraphs.

Interface file of the simulation module adapter

The interface file (.h) of a simulation module adapter is generated by instantiating a port class for each real/logical port.

For instance the following statement in the interface file of the module simulation adapter of the module processor creates an input port of type integer (32 bits):

```
processor.add_port(create_port<sc_in<sc_int<32 > > >);
```

The statement creating an output port on the simulation modules interface is similar to the proceeding one. Thus only the direction (output /input) and the data type will be changed. So an output port of type bit is created by the following statement:

```
processor.add_port (create_port <sc_out<sc_bit > >);
```

add_port and create_port are two methods that we added to our SystemC library in order to simplify and to automate the code generation.

Association rules

These rules are a set of basic primitives allowing performing read/write operations on the ports. Thus generally when a module is specified at transaction level or at RT level the communication is more or less explicitly modeled, which is not the case for modules described at UTF or TF levels. So when we have for example to simulate at RT level together in a system: a module where the communication is explicitly modeled (RTL) and another module at UTF level, and we want to simulate the system at the RT level, we must use these rules to associate these different ports.

The Association rules are some simple rules describing the ports association between the system modules. Thus, for instance a memory described on RTL level communicates via a certain number of control ports and sets of address and data ports, a processor described at level UTF reaches simply a memory with an address port and a data port both of type integer, and possibly some control ports (WriteEnable for example). To simulate at RT level a system containing these two modules, the association rules allow associating the processor data port (integer) with the set of the memory data ports the memory for instance. Assuming that the processor data port is called PDP and the memory ports are respectively called PMD0, PMD1, ..PMD15, the association rules will be expressed simply by the expression:

PPD – PMD0
– PMD1
– PMD2
– …
– PMD15

This means that the ports PMD0 to PMD15 receive their data from the port PPD.
For instance, the following statement is generated from an association rule. It associates an integer (32) real port with a logical (bit vector (32)) one in the simulation module adapters interface.

```
m1.add_trans_in (TRANS_GENE (sc_in<sc_int<32 > >, sc_out <sc_bv<32 > >, sc_bv<32 >, m1[0], m1[1]));
```

TRANS_GENE is a macro defined in file (Association.h). It manipulates the defined association rules.

Behaviors file of the simulation module adapter

The simulation module adapters behavior is an infinite SystemC process sensitive to all (or a set of) its own input ports (real ports). For instance, the following SystemC statement makes m1's behavior sensitive to its input ports:

```
m1.set_process_in ();
```

Where m1 is a simulation module adapter, and set_processe_in() is a defined method making a module sensitive to any changes in its input ports.
After that we may specify a specific set of sensitivity ports to the simulation module adapter. For example the statement:

```
m1.set_event (((sc_in<bool >*)m1[2])->default_event ());
```

makes the module m1 sensitive to its second input port (m1[2]) which is Boolean. So, the behaviors infinite process of m1 will be executed at each new value on this port.

The behavior is then generated using transformation rules.

Transformation rules

As the preceding ones, they are quite simple rules. They mainly define the protocol transformation between the system modules. Thus, in the example of the memory and the processor, a change of value on the processor data port implies a write operation on some memory data ports in a quite precise order (the memory access protocol is provided by the memory vendor).

The following example of a transformation rule means that when the processor writes a data on its data port, three ordered write operations must be performed on the memory ports. Thus, the simulation module adapter must writes data on the memory control ports PMC0 and PMC1 in the same cycle, followed by a write operation on the memory data ports PMD0 to PMD15 in the next cycle and finally a write on the control port PMC3.

PPD – PMC0, PMC1
– PMD0,
– PMD1,.., PMD15
– PMC3

This rule for instance, configures the simulation module interface of the processor module as follow:

```
generic_container * gc = new generic_container;
…
gc.add_trans (TRANS_PUT_VALUE (sc_out<sc_bit>,
0, 1, m1[18]));
gc.add_trans (TRANS_PUT_VALUE (sc_out<sc_bit>,
1, 1, m1[19]));
gc.add_trans (ADD_WAIT (2));
gc.add_trans (TRANS_GENE_SET (sc_in<sc_int<32>
>, sc_out<sc_bv<32> >, 0, 15, m1[0], m1[1]));
gc.add_trans (ADD_WAIT (2));
gc.add_trans (TRANS_PUT_VALUE (sc_out<sc_bit>,
3, 1, m1[21]));
...
m1.add_trans_in (gc);
```

Where **TRANS_PUT_VALUE** is a method allowing to set a giver port, on a fixed value.

Note. For the moment we are specifying the rules simply in a text format, but we are working on a specific formalism and language to define them, in order to make their definition and reuse simpler and easier.

4.2.3. Modules linker

This last stage consists to connect the each generated simulation module adapter with the module with which it is associated, then to connect the different modules together to lead finally to an executable simulation model allowing simulating the whole system.

5. APPLICATION

We want to validate by simulation a SoC containing a transmit module specified at UTF level and an SDRAM (Synchronous Dynamic Random Access Memory) described at RT level. The memory is an existing physical module produced by Micron [Mic] under the reference MT48LC16M16A2. It's a 256 Mbits memory, containing four bancs of 64 Mbits. Each one of the blocs is characterized by 8192 lines and 512 columns. Memory words are 2 bytes.

5.1. System modules specification

In order to simulate this system, we first specify the two modules at their respective abstraction levels (see table 1).

5.1.1. Functionality of the memory

The SDRAM is an existing module and its functionality and composition are specified by its vendor. It is composed by several parts; the most important ones are: the control logic, lines, columns and banc decoders, the input/output registers, mask signals, the refresh counter and the clock signal..

5.1.2. Ports types

The communication interface of the SDRAM is defined by the vendor. Thus, in order to connect it with other modules we must respect exactly this interface (number of ports, ports senses and ports types).

Note. The time needed for each operation on the SDRAM is given by the vendor in the form of a low and up bounds for each one. In our implementation of the memory, we choose the average of the two bounds for each operation time.

5.1.3. SDRAM Commands

To send any command to the SDRAM, the processor must send signal on a set of ports. These signals are specific to each command. The signals associated with each command are given by the vendor. Thus for instance, in order to perform a write operation on the SDRAM, the signal RAS must be set to 1 (H) and the signals CS, CAS and WE to 0 (L).

5.1.3. Transmitter module specification

The transmitter module is a high level module dedicated to communicate with the SDRAM in order to verify its functionality. The transmitter's interface is composed by 5 ports as shown below:

```
sc_out<sc_uint<addr_size>> addressTrans;
sc_in<bool> clock;
sc_out<bool> enable;
sc_out<bool> readwr;
sc_out<int> command;
sc_inout<sc_uint<data_size>>
dataTrans;
```

The command port is used to specify which operation the transmitter wants to perform on the SDRAM. An integer value is associated to each command of the memory.

5.2. Simulation model generation

We choose to simulate our system at RT level. So we generate a simulation module adapter to encapsulate the processor module and allow it to communicate with the SDRAM.

5.2.1. Rules definition

The association rules consist to associate the simulation module adapter's real ports with its logical ports. Thus, all the memory ports are associated with at least on transmitter port. For instance, the following two rules:

ADDRESSTrans – ADDR
- CAS
- RAS
- DMQH
- DMQL
DATATrans – DATA

means that de real port of the transmitter's simulation module adapter **ADDRESSTrans** will be associated with the logical ports ADDR, CAS, RAS, DMQH and DMQL. Thus, these five logical ports will take their values from the specified real port. So for this application we defined one association rule for each transmitter port.

The transformation rules are all those defining the access protocol to the SDRAM as defined by the vendor. Thus, we defined 11 transformation rules (one for each memory command). Each one of these rules defines how and in which order the value(s) on the real ports are transformed to be written on the logical ports.

5.2.2. Simulation module generator

The generated module contains 22 lines in its interface file (.h) and 90 lines describing the infinite process "behavior" (.cc). As shown in the table 1, the generated code has a very acceptable code compared to the code necessary to specify the SDRAM at the RT level.

	SDRAM	Processor	Sim. Mod.
Interface	42	60	22
Behavior	1200	170	90

Table 1: Simulation models modules code size.

5.2.3. Modules linker

During this step, we connect the generated simulation module adapter with the processor module, and then we connect it with the SDRAM module. This consist simply to link point-to-point the simulation adapter ports with those of the two system modules. Thus the modules linker instantiates 3 signals to connect to processor with its simulation module adapter's real ports and 10 signals to connect the SDRAM with the simulation module adapter's logical ports.

5.3. Conclusion and analyses

This application shows that our mythology permits to simulate together an existing industrial module described at RT level (SDRAM) and another module specified at a high abstraction level in a fast and simple way. The generated multi level simulation model has an acceptable complexity compared with the initial specification of the system to be simulated.

6. CONCLUSION AND PERSPECTIVES

In this paper we presented an automatic multilevel simulation methodology for SoC design. Our approach consists to instantiate and to configure a generic C++ class to obtain one simulation module adapter for each module, described at an abstraction level different from the simulation level, in the system. The effectiveness of this approach has been shown on an example composed by an SDRAM and a processor described respectively at RT and system levels.

REFERENCES

[Bru00] J-Y Brunel, W. M. Kruijtzer and al, "COSY Communication IP's", Proc. of the DAC, Los Angeles, CA, June 200.

[Cho95] P. H. Chou, R. B. Ortega and al. "The Chinook Hardware/Software Co-Synthesis System", Proc. of the ISSS, 1995.

[Cow02] Coware. Inc., "N2C" available at http://www.coware.com/cowareN2C.html

[Gro02] T. Grotker, S. Liao and al, "System Design with SystemC", Kluwer Academic Publishers, USA 2002.

[ITR00] International Technology Roadmap for Semiconductors.

[Mic] MICRON, http:\www.micron.com

[Mef02] S. Meftali, "Architectures exploration and memory allocation/assignment for multiprocessor SoC", PhD thesis, University Joseph Fourier, TIMA laboratory, France, 2002.

[Nic02] G. Nicolescu, "Specification and validation of heterogeneous embedded systems", PhD thesis, INPG, TIMA laboratory, France, 2002.

[Sem00] L. Semeria and A. Ghosh, "Methodology for Hardware/Software Co-verification in C/C++", Proc. of the ASPDAC, Jan. 2002.

[Sys] SystemC language and user guide, available at http://www.systemc.org

[Vsi] http://www.vsi.org/

Automatic Generation of, Geographically Distributed, SystemC Simulation Models for IP/SoC Design

Author(s) Name(s)
Author Affiliation(s)
E-mail

Abstract

In this paper, we present a methodology permitting automatic generation of, geographically distributed, SystemC simulation models for IP/SoC design.

We construct the SystemC simulation model as a client/server application, where the clients are the IPs composing the system. The server in a SystemC synchronization module, it drives the simulation process.

The communication between SystemC simulators residing on distant hosts is realized by using sockets.

The effectiveness of our approach is shown on a version of a packet rooting switch composed by four processors and one shard memory.

1. Introduction

Now a days SoCs are more and more complex and integrate software parts as well as specific hardware parts (IPs "Intellectual Properties" generally). Due to this complexity, the time spent to verify and validate these systems is continuously increasing and may be usually greater than half of the time-to-market of a system.

When communication means become increasingly powerful, it becomes more and more interesting for designers to have tools allowing them to simulate systems on distributed platforms. Thus, a user can simulate his complete system on geographically distant hosts. This functionality makes it possible to different design teams to cooperate for the simulation and thus for the design. This type of simulation allows a broader use of competences and intellectual resources by specialized teams.

This possibility of simulation makes it possible to designers to avoid the licences problems. Distributed simulation does not require to have the licences and the simulators on the same machine. It allows contrary to distributing the engines of simulation in the specific groups generally having the licences for the dedicated simulators. It is possible thanks to this method to combine the machines and the software in order to share available CPU resources for the various simulated subsystems, in order to speed up the hole system simulation. Moreover, geographically distributed simulation makes it possible to verify a system by considering its different modules (component) as black boxes residing on the hosts of their designers (providers). This completely meets the need for the reuse in the SoC design. Indeed, the distributed simulation allows to simulate a SoC designed by assembling IPs having just their interfaces and without needing to download their behavior on a local machine.

These last years have been marked by the appearance of SystemC [4] which becomes more and more used for the SoC design. The beginning of development of reusable components (IP) in SystemC makes that the need for tools allowing distributed simulation of SoC described in SystemC is increasingly pressing. Unfortunately, such a tool does not exist yet, so for this reason the development of such a tool is the main goal of our current work.

This paper is organized as follow: in section 2, we present some necessary basics to understand our methodology. In section 3 we give a review of state of the art on distributed simulation. The different steps of our methodology are presented with details in section 4. The use of our approach is detailed on an example in section 5. Our approach is applied to a version of a packet rooting switch in the section 6. We conclude this paper in the section 7.

2. Basics

We mean by a distributed simulation an executable model of a given system on two (or more) concurrent simulators running on distant hosts. Thus, each module or IP (processor, memory,…etc) of the SoC may be executed on a given host as shown in the Figure 1. For some commercial or licenses constraints, some IPs must be executed on a given host and the designer has all the freedom for choosing the hosts (from the available hosts) on which the remaining IPs will be executed.

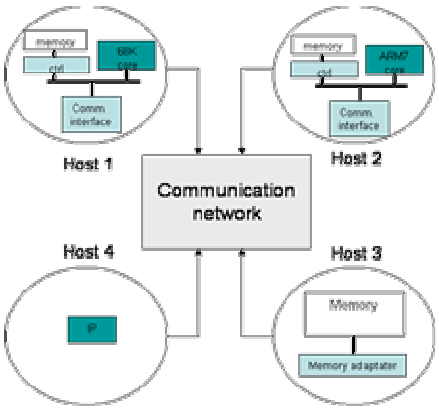


Figure 1. Distributed simulation on distant hosts.

3. Related work

We find in the literature several tools allowing the geographically distributed simulation of SoC. But only Plug&Sim [3] permits distributing SystemC. Unfortunately in this tool we can't use more than two SystemC simulators. A more or less exhaustive presentation of all these work is given in [2].

4. Methodology

Our distributed simulation models generation is mainly composed by two steps. Thus, starting from the communication interfaces description of the SystemC modules composing the application, we find an optimal assignment of these modules on the available hosts for the simulation. This step is described in details in [1]. The second step consists in transforming the initial application code in order to adapt it to the distributed simulation platform find in the previous step. This transformation step is described in this section.

4.1. SystemC and networks

The standard SystemC API don't allow distributed simulations on a network. Thus, in order to make it possible, we use sockets standard API instead of SystemC one. we never modify this later in order to make sure that our methodology can work on any SystemC version or platform.

4.2. SystemC signals

To permit the communication between IPs in SystemC, we frequently use sc_signal. This is a standard solution for simulations on a single host. But, as our goal is to realize multi-hosts simulation, we defined the class sc_signal_socket which is similar to sc_signal but where the read/write methods have been redefined. An example of declaring and using such signals is given below.

Client_connection c1(arg, argv) // allows the communications management
IP_A m1 ; // instantiating an IP
sc_signal_socket<int> s1 (&c1, 4578); // creating a signal
m1.out_data (s1); // bind the created signal to a port

Note. The operation of binding sc_signa_socket to ports is exactly similar to sc_signal one.

4.3. Communications and interconnections management

Our distributed simulation models are closely similar to Client/Server applications. Thus, our approach consists in creating a server able to store and dispatch information received from the clients.

4.4. Server side

The server is a SystemC process permitting to synchronize the clients. Thus, for each simulation, we must start running the server with arguments containing some information about the clients. These arguments are:

- the number of the clients participating to the simulation (it corresponds to the number of IPs in the SoC)
- simulation time
- simulation step
- communication port (port connecting the client's host to the server's one)

We can see the syntax of this operation on the following example:

#> ./server -n 2 -d 400 -u NS -p 8566

When the server starts running, it waits for the clients. After the connection to all the clients to the server, this later sends to each one of them the simulation parameters and acknowledge. This permits to each client to start itself simulation. During the simulation, clients perform read/write operations via their connections to the server.

In order to synchronize the simulation and ensure the maintaining of application's semantic, the server uses handshake protocol as follow:

- When a client performs a write operation, it will not be accepted by the server unless the last data has been read by its owner.
- When a client performs a read operation, the server sends the needed data if and only if it has a new value.

The server stops running when all clients end their simulation.

Note. It is possible to improve the performances by adding FIFOs on the ports connecting the clients to the server

4.5. Client side

The client functioning is simpler than the servers one. Indeed, to start running a client, it is enough to give to him the IP address and the communication port of the servers host as parameters. This is shown on the following example:

./client_IPA -a 134.251.23.12 -p 8566

Thus, the client connects to the server, receives the simulation parameters then waits for the server's acknowledge to start itself simulation. After the simulation of the system has been started, the communications (read/write) via the sc_signal_socket signals are directly submitted to the server.

Note. In order to permit to the sever to recognize and make the correspondence between signals of a client A and a client B, we associate to each signal a given single number.

5. Example

To illustrate our distributed simulation approach, we will take an example of a system composed by a producer

and a consumer. The producer and the consumer are two IPs described in SystemC. The needed code to adapt these IPs to a two hosts simulation platform (one SystemC simulator running on each host) is very simple as shows on the following code.

5.1. Producer's code

```
int main (int argc, int argv)
{
client_connection c1 (argc, argv); // permits the
communication management
sc_produce p ("produce");  // the IP producer
sc_signal_socket<int> s1 (&c1, 24);  // creating a signal
p.out_data (s1);   // binding the signal to the producer's
data port
sc_clock CLK ("clock_1"); // declaring a clock
p.clock (CLK); //
c1.get_synchro (); // synchronization with the server
sc_start (c1.get_sim_params ()); // starting the
simulation with the parameters received from the server.
return 0;
}
```

5.2. Consumer's code

```
int main (int argc, int argv)
{
client_connection c1 (argc, argv); // permits the
communication management
sc_consum p ("consum");  // the IP consumer
sc_signal_socket<int> s1 (&c1, 24);  // creating a signal
p.in_data (s1);   // binding the signal to the consumer's
data port
sc_clock CLK ("clock_1"); // // declaring a clock
p.clock (CLK);
c1.get_synchro (); // synchronization with the server
sc_start (c1.get_sim_params ());// starting the
simulation with the parameters received from the server.
return 0;
}
```

Note. The code corresponding to the communication of these two clients supposes only the knowledge of their respective communication interfaces.

5.3. Running the server

For this simulation, we have 2 clients (one for each module of the system). The simulation will be performed for 400 ns on the port 8566 of the IP 134.252.16.31. So the following code starts the server's simulation.

```
#> ./server -n 2 -d 400 -u NS -p 8566
```

5.4. Running the clients

On the host A we simulate the producer as follow:

```
# ./client_producer -a 134.252.16.31 -p 8566
```

On the host B we simulate the consumer as follow:

```
# ./client_consumer -a 134.252.16.31 -p 8566
```

6. Application

In order to illustrate the efficiency of the proposed methodology on a real application, we applied it to a packet rooting switch [1].

It constitutes a powerful solution for large-frame or cell-switching systems. The version we present here consists of two input controllers and two output controllers. Each of the controllers handles one communication channel. The communication links between input and output controllers are configured by an external signal to be direct or switched. Figure 2 shows the block diagram of the packet routing switch.

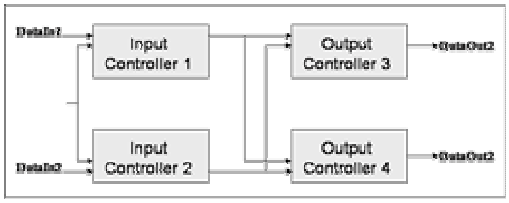


Figure 2. Block diagram of the packet routing

The results of this experiment have been very satisfying. The details are giver in [2]

7. Conclusion

In this paper, we presented a new approach for SystemC distributed simulation. It is a multi-host simulation for IP based SoCs described using SystemC. All the necessary classes to use SystemC on a network have been defined and implemented, this makes our methodology systematic and completely automatic.

In this approach, the system specification is considered as a client/server application, where each IP is a client.

The simplicity and the effectiveness of the methodology have been shown on a producer/consumer example.

7. References

[1] IBM Inc. "28.4G Packet Rooting Switch", Networking Technology Data sheets, http://www.chips.ibm.com/techlib/products/commun/datasheets.html

[2] S. Meftali, Jean-Luc Dekeyser, "An Optimal Charge Balancing Model for Fast Distributed SystemC Simulation in IP/SoC Design", Research report, LIFL, June 2003.

[3 OMG-CORBA, http://www.omg.org

[4] SystemC user's manual, www.systemc.org.

An MPSoC Performance Estimation Framework Using Transaction Level Modeling

Rabie Ben Atitallah Smail Niar Samy Meftali Jean-Luc Dekeyser
INRIA-FUTURS, DaRT Project
Laboratoire d'Informatique Fondamentale de Lille
University of Lille, France

E-mail: {benatita, niar, meftali, dekeyser}@lifl.fr

Abstract

To use the tremendous hardware resources available in next generation MultiProcessor Systems-on-Chip (MPSoC) efficiently, rapid and accurate design space exploration (DSE) methods are needed to evaluate the different design alternatives. In this paper, we present a framework that makes fast simulation and performance evaluation of MPSoC possible early in the design flow, thus reducing the time-to-market. In this framework and within the Transaction Level Modeling (TLM) approach, we present a new definition of the timed Programmer's View (PVT) level by introducing two complementary modeling sublevels. The first one, PVT Transaction Accurate (PVT-TA), offers a high simulation speedup factor over the Cycle Accurate Bit Accurate (CABA) level modeling. The second one, PVT Event Accurate (PVT-EA), provides a better accuracy with a still acceptable speedup factor. An MPSoC platform has been developed using these two sublevels including performance estimation models. Simulation results show that the combination of these two sublevels gives a high simulation speedup factor of up to 18 with a negligible performance estimation error margin.

1 Introduction

According to Moore's law, more and more transistors will be integrated on a single chip. Such a huge transistor budget makes it increasingly difficult for engineers to design and verify the very complex chips that result, and in turn widens the gap between silicon capacity and design productivity. MultiProcessor Systems-on-Chip (MPSoC) architecture has thus become a solution for designing embedded systems dedicated to applications that require intensive computations. The most important design challenges in such systems consist in reducing simulation time and estimating performance appropriately.

Traditional approaches for performance estimation at the Register Transfer Level (RTL) cannot adequately support the level of complexity needed for future MPSoC, since RTL tools require great quantities of simulation time to explore the huge architectural solution space. Recently, significant research efforts have been expended to evaluate MPSoC architectures at the CABA (Cycle Accurate Bit Accurate) level [21] [3] in an attempt to reduce simulation time. Usually, to move from the RTL to the CABA level, hardware implementation details are hidden from the processing part of the system, while preserving system behavior at the clock cycle level. Though using the CABA level has allowed accurate performance estimation, MPSoC DSE at this level is not yet sufficiently rapid compared to RTL [2].

In our work, we focus on the use of Transaction Level Modeling (TLM) in an MPSoC design which corresponds to a set of abstraction levels that simplifies the description of inter-module communication transactions using objects and channels between the communicating modules [10]. Consequently, modeling MPSoC architectures becomes easier and faster than at the CABA level. Unfortunately, little research has been devoted to propose efficient MPSoC simulation models for reliable DSE. In addition, the proposed frameworks suffer from accurate performance estimation tools appropriate to nowadays complex MPSoC architectures such as hierarchical and distributed systems.

As our objective in this paper is to propose reliable environment for rapid MPSoC design space exploration, the framework has been designed in the context of PVT level [8]. In the conventional definition of the PVT level, the hardware architecture is specified for both processing and communication parts, as well as some arbitration of the communication infrastructure is applied. In addition for performance estimation, this level is annotated with timing specification. Using a "top-down" approach, we propose a PVT framework composed of two sublevels: PVT Transaction Accu-

13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications(RTCSA 2007)
0-7695-2975-5/07 $25.00 © 2007 IEEE

rate (PVT-TA) and PVT Event Accurate (PVT-EA). PVT-TA operates at a relatively high abstraction level and does not take a specific communication protocol into account. This permits a rapid exploration of a large solution space by eliminating non-interesting regions from the DSE process. Solutions selected at the PVT-TA sublevel are then forwarded for a new exploration at the PVT-EA sublevel. This second sublevel specifies a precise communication protocol and takes architectural delays into account. Because estimation methodology that we developed for the PVT-EA is more accurate, it is possible at the price of less simulation speed, to locate the most efficient architecture configurations. PVT-TA and PVT-EA permit the use of PVT models in a coherent methodology, and to have accurate estimations.

In this paper, we make several contributions. First, we define two new complementary sublevels in the PVT level that allow different trade-offs between speed and accuracy. Second, we define a refinement phase that allows the simulation to move from the highest sublevel (PVT-TA) to the lowest (PVT-EA) sublevel. Third, using these two sublevels, we have developed a multiprocessor platform including performance estimation tools suitable for each sublevel.

The rest of this paper is organized as follows. An overview of related work on existing simulation speedup techniques at TLM for MPSoC is provided in section 2. Our PVT framework for MPSoC modeling is introduced in section 3. Details about the MPSoC component models at the PVT level are presented in section 4, and our performance estimation model is described in section 5. Section 6 presents the experimental results obtained when applying the proposed framework for an MPSoC DSE and a set of parallel applications. Finally, section 7 gives our conclusions and prospects for future research.

2 Related Work

During last years, a lot of research on DSE for embedded systems have been conducted. As a result of these research, several exploration environments are proposed, such as MILAN [19], GRACE++ [12] and Metropolis [1]. The work presented in this paper could be considered as complementary to these environments. Our PVT-TA and PVT-EA multilevel simulation tool with its accurate performance estimation could be easily integrated in such frameworks. SystemC is a framework where a homogeneous co-simulation (software and hardware) can be realized. Compared to traditional heterogeneous co-simulation tools, SystemC (and especially TLM) makes simulation faster, as the abstraction level has been elevated from RTL to transaction level. Moreover, overheads associated with synchronization and communications between simulation models can be greatly reduced with TLM. Since the first proposition of TLM in 2000 [10] [13], an increasing number of research projects have considered the problem of its definition, which has led to a multitude of different frameworks [8] [6] [4] [9]. All of these researches have two factors in common: 1) TLM's are presented as stacks of several levels and 2) the communication and computation aspects of the frameworks are kept separate. However, these frameworks lack tools for fast performance estimation at high abstraction modeling levels, especially for MPSoC platforms.

To our knowledge, few studies have been devoted to these requirements in the past. Among the approaches that are similar to ours, we cite those proposed by Pasrisha et al. [20], Viaud et al. [22] and Kim et al. [14]. In their paper, Pasrisha et al. presented a new abstraction level on top of RTL. Their proposed Cycle Count Accurate at Transaction Boundaries (CCATB) level maintains the same degree of accuracy as RTL. The CCATB level has some points in common with our PVT-EA. In CCATB, the timed TLM model has also been enhanced with communication protocol control and timing information. Still, our PVT-EA is different from CCATB as it is designed for MPSoC platforms with any type of interconnection network, not necessarily a shared bus. The speed up they obtained is smaller than ours.

Viaud et al. have proposed an ambitious timed TLM based on conservative parallel discrete event theory. They obtained a high speedup simulation factor but they did not measure this speedup on real applications. Their model is also different from ours. Firstly, our approach can be applied for hierarchical or distributed MPSoC design, and secondly, they used "null messages" to read master clocks which can increase the communication overheads.

Kim et al. [14] proposes a new technique for HW/SW co-simulation for heterogeneous MPSoC platforms. The basic idea behind this paper is to combine an efficient trace driven simulation and virtual synchronization. In traditional trace driven simulation the application is first executed and only the events that need to be simulated are recorded (trace generation). In the second step, collected traces in the first step fed an ISS to realize events alignment thus allowing performance evaluation. As in [20], only bus based MPSoC platforms are considered. In addition, the approach we present in this paper is different as it is based on TLM and does not need disk space to store traces.

3 MPSoC Modeling at PVT Level

This section describes in detail the two sublevels composing our PVT framework and introduces the refinement phase that allows the model to be transformed automatically from the first sublevel to the second. Fig 1 summarizes our PVT framework proposal.

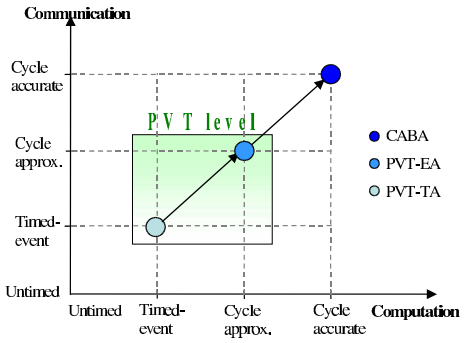


Figure 1. MPSoC modeling for DSE

3.1 Transaction Accurate PVT (PVT-TA)

Two objectives have been defined for our PVT-TA sublevel. Firstly, fast verification of system functionalities and secondly dynamic contention monitoring in the interconnection network in one side and the corresponding timing information in the other side. In fact, the second objective makes the difference between PVT-TA and the standard Programmer's View (PV) level which does not include timing specification. Simulation results at the PVT-TA sublevel allow taking decisions such as: the adequacy of the interconnection network with the application needs, the mapping of tasks on the processors and the mapping of data into memory banks. MPSoC description with PVT-TA produces the same behavior at the transaction level as with CABA, thus, the derived name "Transaction Accurate". In PVT-TA, details related to the computation resources, such as the cache FSM or the processor control unit, are omitted. Details related to communication are also hidden. To do so, transactions are performed through channels [13] instead of signals as used at the CABA level. The channels implement one or several interfaces, and each interface has a set of read and write methods. To load or store data, read() or write() function calls are instantiated by masters and sent through the port to the channel interface. At the level of slaves, the transaction will be recovered to execute the corresponding methods and to send the response.
In this sublevel, a timing model is defined (section 5.1) to approximate the execution time. Nevertheless, there is no defined communication protocol. Events occur successively with no delay between them. As a result, system behavior changes, which may alter the accuracy of performance estimation.

3.2 Event Accurate PVT (PVT-EA)

The objective of the PVT-EA sublevel is to explore selected solutions at the PVT-TA sublevel and to minimize the lack of accuracy inherent to this sublevel. To do so, we identified the communication and the processing details that must be added into the model to increase the accuracy without significantly diminishing the simulation speedup. Among all the architectural details, the awareness of the communication protocol and the respect of delays between microarchitectural events demonstrated the most influence on performance estimation accuracy. This can be explained by the fact that the simulated MPSoC is able to execute several tasks concurrently. To ensure correct ordering of events between processors, it is thus necessary to pay particular attention to the execution delays (instruction execution, request and response packet preparation, data request transmission via the network, etc). This attention to delays between instructions allows establishing the same event sequencing obtained at the CABA level.
At this sublevel, a precise communication protocol must be specified by the user. Moreover, architectural delays (such as functional units and network response time) must be given. These delays can be either approximated by the user or mesured from a lower level. Moving from the PVT-TA sublevel to the PVT-EA sublevel occurs automatically using a refinement that adds timing and protocol details to the PVT-TA MPSoC description. These details corresponds, for example, to wait() instructions with arguments determined by the platform configuration. Table 2 presents delays of the platform, expressed in number of cycles, that have been used in the experiments.

4 Component Models for MPSoC Design

In this section, architectural component models for MPSoC design are presented. These models are generic and allow to evaluate the performance of the proposed implementation of PVT. The component models that have been designed in our framework are: processors, caches, interconnection network, RAM, DMA controller and a DCT hardware accelerator.
Using SystemC version 2.1 and the TLM library, each of these component models have been implemented at the PVT-TA and PVT-EA sublevels. Even if the description of these models is linked to a specific SystemC implementation, we think that they are essential to understand the behavior of each component at run-time. The difference between the PVT-TA and the PVT-EA for these models is in the corresponding timing specifications as explained in the next section. In the following, we give details concerning the models of components used in section 6.

4.1 The Processor model

With a TLM approach, two major abstraction levels are used for the processor: the cycle-accurate level and the instruction-accurate level. At the cycle-accurate level,

the micro-architectural details, such as pipeline stalls and branch prediction penalties, are simulated, which allows a very accurate model to be obtained. At the instruction accurate level, the processor simulator executes instructions sequentially and micro-architectural details are approximated by assigning average delay penalties to instructions. Timing annotations are added into the simulator to provide accuracy close to that of the cycle-accurate level. In both TA and EA sublevels of our PVT framework, instruction accurate simulator has been chosen to allow an interesting simulation speed. Moreover, the processor functionality was inserted into a thread and the processor interface was modified in order to replace the signal interface with a channel interface. This modification increases the simulation speedup. Two methods, of type read() and write(), have been defined to permit to processor to instantiate function calls for data and instruction loading. Our platform uses scaler in-order MIPS R3000 processors with two ports for a direct communication with data and instruction caches via blocking channels.

4.2 The Cache memory model

Usually [11] [22], the cache model is integrated in the processor component. In the opposite, in our framework, the cache model is standalone. This design modularity allows different cache architectures to be explored and evaluated with the same processor core. The data and instruction caches in our framework are modeled with one slave port to exchange requests and responses with the attached processor and one initiator port connected to the interconnection network to instantiate the read() or write() function calls. The latter is used in the case of cache misses. To avoid the problem of cache coherency, each data segment is divided into two parts: ”cached” data and ”non-cached” data. ”Cached” data part contains local data while ”non-cached” part contains global data. Thus, when a data cache miss happens, the corresponding data block is loaded from the memory component only if the block belongs to a ”cached” part.

4.3 The Interconnect Model

The function of the interconnection network is to route the traffic generated by initiators (i.e. processors, caches) to the different targets (i.e. memory and I/O peripherals). Though it would be valid for a variety of interconnection topologies (e.g., shared bus with arbiter, multistage interconnection network, mesh), in the experiments presented in section 6, the modeled interconnection network is the crossbar. This crossbar is composed of two types of modules: the router and the arbiter. The router is able to implement several protocols (VCI, OCP, etc.) and uses an address map table. It contains several ports that can be connected via channels to several slaves. The address map table used by the router contains a set of address ranges assigned to each channel. The router was implemented using a simple, yet efficient, C++ function instead of a SystemC thread.
An arbiter is essential to handel conflicts between possible simultaneous requests for the same target. When the master needs to access a shared module via the network, it puts a request into the corresponding channel and waits for a response. At the arbiter level, a thread reads requests from the FIFO channels. Using a round-robin arbitration policy, the arbiter decides which master has to serve next, and forwards the selected request to the slave. When the slave responds, the arbiter puts the response into the response FIFO on the relevant channels. The master then picks up the response and completes the transaction. Fig 2 illustrates how a 2*2 crossbar might be implemented. Though, the presented interconnection model is simple, it is however sufficient to obtain the defined objective of dynamic contention observation. Such a 2*2 crossbar makes it easier to design other types of interconnection networks, such as a multistage network.

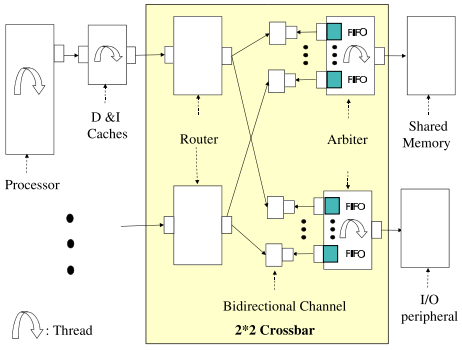


Figure 2. 2*2 Crossbar implementation

4.4 The Memory model

The memory module is a slave component and includes two SystemC methods: read() and write(). Using the sc_export mechanism introduced in SystemC 2.1, these two methods are called directly by the master and thus executed in the master context. Consequently, there is no need for a separate thread in the target. This sc_export mechanism is used to model all the slaves in our MPSoC platform (e.g., timer, locks engine). In our PVT framework, the slave port is either connected directly to the master's router if the target is private (e.g., caches) or is connected to the arbiter module if the target is shared between the processors.

4.5 The Direct Memory Access Controller (DMAC) Model

This DMAC device can be considered either as a target that can be configured by a processor or as an initiator that realizes transfers of data blocks from a source address to a destination address in the data memory module. This component is modeled with two initiator ports to exchange requests and responses with the corresponding source and destination target ports and one target port is used in the case of DMAC configuration.

4.6 The 2D DCT Hardware Accelerator Model

The 2D DCT coprocessor component allows the transformation of an 8x8 image block from the spatial domain to the DCT domain [15]. The 2D DCT is implemented by two 1D DCT's. The first one operates on the eight rows while the second one operates on the eight columns. Several fast algorithms are available for the calculation of the 8-point 1D DCT. In our coder, a modified version of Loeffler algorithm has been used [17]. This method has been selected due to the minimum number of required additions and multiplications (11 multiplications and 29 additions). The algorithm has been tested on the Altera STRATIX EP1S10 development boardčitealtera. Implementation result shows that the 2D DCT coprocessor frequency can reach 120MHz. In addition, a timing analysis is performed to deduce the activities delay in this component; these values are added later in the performance model for accurate estimation in the PVT sublevels. At the transactional level, the 2D DCT coprocessor is presented as a target answering to processor requests.

5 Performance Estimation

This section illustrates the utilization of the component models, previously presented, for MPSoC performance estimations at the two PVT sublevels (i.e. TA and EA).

5.1 Performance estimation with PVT-TA

The timing model is integrated in our sublevels as a plug-in, which facilitates accurate performance estimation for DSE. The proposed performance estimation methodology should be flexible to be adaptable to different architecture topologies such as memory, distributed memory and hierarchical systems. In addition, the proposed solution should take into account some timing issues such as those linked with the processor's synchronization, dynamic contention in the interconnect and communication protocol specifications.

Our strategy is based on identifying for each component the corresponding pertinent activities, such as the number and types of executed instructions for the MIPS processor; hits and misses for the caches; the number of transmitted/received packets for the interconnect; the number of read and write operations for the shared memory modules; the number of data transfers done by the DMAC, etc. A counter, incremented during simulation, is attributed to each type of activity. In addition to counting the activities, execution time estimation requires attributing a time delay value to each type of activity. In our approach, execution time delays are either measured from a physical characterization of the hardware component or from an analytical model. In our experiments in section 6, the two approaches are used. For instance, MIPS processor instruction execution delays are estimated from the technical document given by [18]. For the cache memory, access delays are obtained using the analytical model proposed by the CACTI tool [5]. Finally, wire delays in the interconnect network are derived from the analytical model described in [2]. Fig 3 shows the utilization of activity delays are used to approximate execution time at the PVT-TA. Symbols used in this figure are described in table 1. A local time counter is attributed to each processor (called local_timer in the figure), and its value is incremented after the execution of each instruction (Fig 3, step 0). Here Op_add corresponds to an execution of an add instruction. The local time counter is then incremented by the instruction execution delay. When an instruction or data read request comes to the processor, a function call to the instruction or the date cache is made. This call is symbolized by read (adr, data, time) instruction (Fig 3, step 1). The time parameter is used to measure the time elapsed between the sending of the request and arrival of the response. It will be recovered by the processor to increment its own local time counter. In the case of data cache hit, the elapsed time corresponds to the cache read time (Fig 3, step 2). In the case of a data cache miss, the cache initializes a new request that is transmitted via the interconnect to the corresponding target (Fig 3, step 3). The elapsed time equals the sum of the transmission time, the cache read (or write) time, and the memory access time (Fig 3, step 4).

The problem is that the transmission time is not constant; its value depends on the dynamic workload. Interconnection network contentions for instance may change its value. To compensate for the lack of constancy, a counter is allotted to each FIFO input buffer on the arbiter side of the system. Therefore, before servicing the selected FIFO, the arbiter polls all the others FIFOs to determine whether or not there is a waiting request. If so, the associated counter (con_trans_word) is incremented (Fig 3, step 5), which allows the request waiting time at the interconnect level to be evaluated. Later on, when a FIFO is selected, the counter value is read and multiplied by the time unit (t_word) to transmit one data word. The counter is then reset to zero

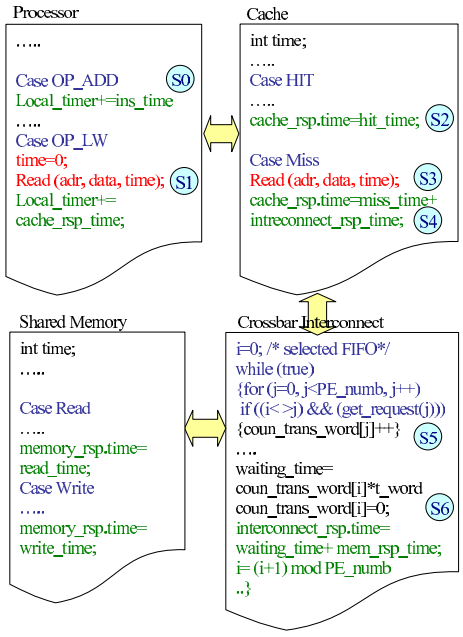


Figure 3. Timing model for performance estimation

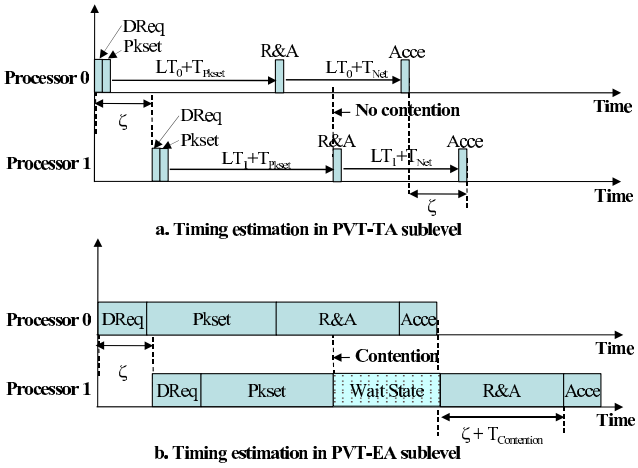
for the next request (Fig 3, step 6). The same approach is used when a DMAC is used. In this case, after initializing the different DMA controller registers, the processor stays in the idle state waiting for a DMAC interrupt. During this time interval, the processor local time counter is not incremented. To recover the time elapsed for the DMA transfer, the DMAC local time counter value is sent to the processor to increment its own local time counter.

5.2 Performance estimation with PVT-EA

The disadvantage of the PVT-TA level is that it does not take the event timing relationship into account. In the case of cache misses in 2 different processors, the time at which a given processor executes the corresponding memory instruction may influence the execution of the memory instruction in the other processor due to collision delays. The Fig 4.a gives an example of contention detection error in the interconnect due to non respecting of event delays (Packet setup, Routing and Arbitration). As it will be demonstrated in section 6, this drawback alters the system's communication behavior, thus reducing the accuracy of the performance estimation. To solve this problem, we enhanced the PVT-TA model with synchronization instructions, incorporating the respect for component activity delays, request/response packet transmission delays and finally communication protocol into our second sublevel PVT-EA. Thus, at this sublevel, events (e.g. misses, instruction execution, interconnection network collisions) occur as they do at the CABA level (Fig 4.b). Though it is true that adding these details implies the specification of a communication protocol, our performance estimation model is so flexible that a variety of communication protocol can be implemented with it.

For comparison purpose, the implementation of the Virtual Component Interface [23] communication protocol used with the CABA platform is described in detail below. In the VCI protocol, data are transferred through packets containing several data words. These packets are generated by the state machine of the communication interface and require several cycles depending on the data block size. Thus, a complete cache line can be loaded in one transaction. Similarly, response packets also require several cycles to be transferred.

To imitate the same protocol behavior at the TLM-EA sublevel, SystemC wait() calls were added before the request and response calls. Likewise, in order to respect component behavior, wait() calls have been added to represents non-zero-delay processor and cache activities, such as instruction executions or cache hits. Table II presents all the delays that have been added to models at this sublevel.



LT0: Local Timer 0; LT1: Local Timer 1; DReq: Data Request
Pkset: Packet Setup; R&A: Routing and Arbitration; Acce: Memory Access
T_{Pkset}: Packet Setup time; T_{Net}: Network time

Figure 4. Timing estimation in PVT-TA and PVT-EA sublevels

6 Simulation Results

6.1 The simulation framework

In order to evaluate the usefulness of the proposed PVT and its two sublevels for MPSoC DSE, several experiments

Table 1. Field Description

Field	description
ins_time	instruction delay
hit_time	cache hit delay
miss_time	cache miss delay
waiting_time	interconnect delay
read_time	read memory access delay
write_time	write memory access delay

Table 2. Architectural delays

Activities	delays
VCI request packet setup	4 cycles
VCI response packet setup	4 cycles
instruction execution	1 cycle
data cache hit	1 cycle
data cache miss	12 cycles + cycles due to contention
read memory access	2 cycles
write memory access	2 cycles
DMAC access	2 cycles
1D DCT to a 8x8 block	12 cycles

were conducted. The aim here was to measure, for each sublevel, the simulation speedup, the accuracy of the performance predictions, and the effort needed to model a given MPSoC architecture. These three metrics have been reported in comparison to the CABA level [21] and are used to illustrate the performance of our approach in DSE for two MPSoC parameters: data and instruction (D&I) cache size and number of processors. Please note that this environment can be used to perform a DSE of a multitude of other architectural parameters (e.g. processor type, NoC type, type of shared memory) in a reasonable time interval with a good level of accuracy. Fig 5 shows the general architecture of the MPSoC used in the experiments. To validate our modeling approach, we parallelized and tested several data intensive software applications: matrix multiplications, high definition downscaler[16], software DCT and H263 decoder [7]. All of these applications have been parallelized by dividing the workload between processors. For instance, the software DCT was parallelized by attributing a frame segment to each processor.
The simulation speedup factor is defined here by the following formula:

$$Speed\_fact = PVT\ Simul.time/CABA\ Simul.time$$

In this formula, PVT corresponds either to PVT-TA or to PVT-EA. For all the experiments, a computer equipped with a 1.6 GHz Pentium M processor and 1GB DRAM is used. When simulations are done at the CABA level, each test requires a simulation time of several hours, depending on the MPSoC configuration.

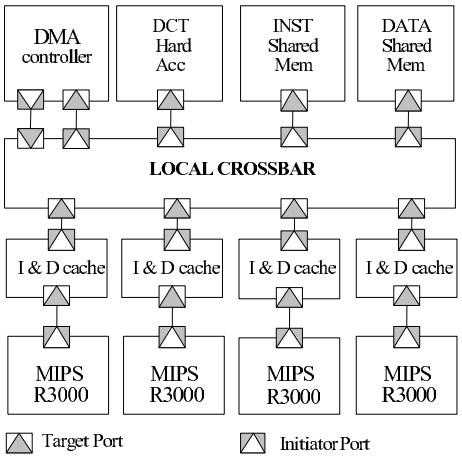


Figure 5. Exemple of MPSoC structure

6.2 Simulation results using PVT-TA

To evaluate the impact of the instruction and data cache size on the simulation speedup factor, we executed our applications on a MPSoC simulator equipped with 4 processors at the PVT-TA sublevel. Instruction and data cache size varied from 256 bytes (B) to 32 Kbytes (KB). Fig 6 shows that reducing the data and instruction cache size improves the simulation speedup factor. This is because reduced cache sizes increase cache misses and consequently the traffic in the interconnection network. This result proves the usefulness of our PVT-TA sublevel for high communicating applications. In addition, we notice an important gap in speedup when the kernel of the application and the corresponding data can not be totally stored in caches. For instance, an important gap in speedup for the H263 application is obtained from cache sizes lower than 16KB. While for the software DCT application, this gap is detected from cache sizes lower than 2KB.

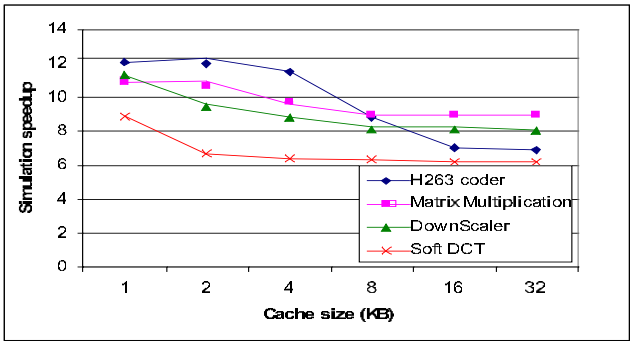


Figure 6. Simulation speedup in PVT-TA for several applications

To evaluate the impact of the number of processors on the simulation speedup factor, we executed the software DCT with different configurations of processor number (from 4 up to 16 processors) at the PVT-TA sublevel. The instruction and data cache size varied from 256B to 32KB.

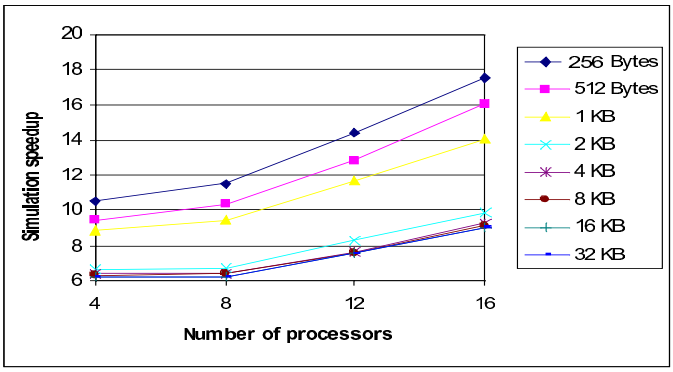


Figure 7. Simulation speedup in PVT-TA for the DCT application with 4 processors

Fig 7 demonstrates that PVT-TA makes it possible to accelerate the simulation by a factor of up to 18. Second, adding processors increased this speedup factor due to the amplification of communication between the processors and the memory modules.

Up to this point, we have focused on the usefulness of our PVT-TA sublevel for performing functional validations with an interesting simulation speed factor. This sublevel may also be useful for dynamic contention monitoring. In this way, it is possible to detect the moment at which a communication bottleneck occurs in the interconnection network. Fig 8 presents the total number of cumulated contentions during the execution of the application. This result is obtained from the soft DCT executed on a platform with 4 processors and 4KB cache sizes. From this figure, the following remarks can be made: first, CABA and PVT-TA curves are close to each other. Second, this figure allows detecting moments at which a communication bottleneck occurs in the interconnection network. In our case, there is an important increase of contention up to 20000 cycles. Beyond this value the amplification is less important.

To evaluate the speedup gain due to the processing part only, we rewrote the DCT application to avoid data cache misses entirely and to reduce instruction cache misses to a negligible number. This application is executed using PVT-TA on our platform, the number of processors was varied from 4 to 16 with 32KB caches. The speedup factor obtained was independent of the number of processors and, more importantly, remained constant (close to 5.4). The results of this experiment confirm that the minimal speedup factor for this application at the PVT-TA level is close to 5.4. Consequently, the rest of gain obtained from this sub-

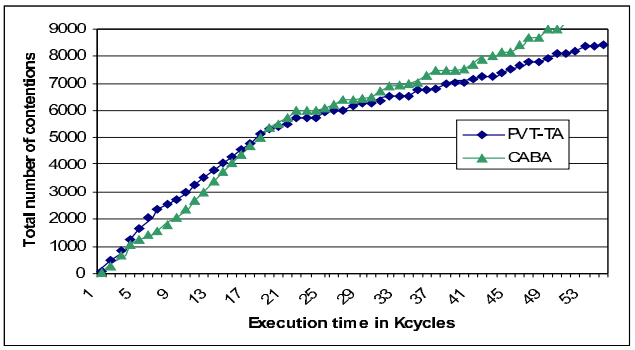


Figure 8. Interconnect contentions in PVT-TA and CABA for the DCT application

level is due to the communication part.

Despite its performance in terms of speedup, the PVT-TA sublevel suffers from a significant performance estimation error. As shown in Fig 9, when the number of processors increases or when the cache size decreases, communication becomes more significant and estimation error increases. This error can be as much as 28% for 16 processors and 256B cache size. As explained in section 5, this is because the MPSoC behaves differently with PVT-TA in terms of the contentions in the interconnection network.

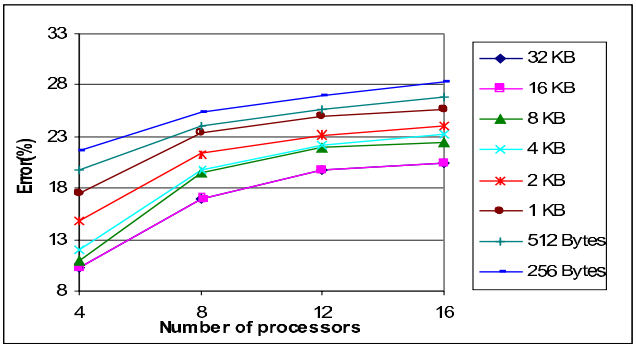


Figure 9. Estimation error in PVT-TA

6.3 Simulation results using PVT-EA

Several experiments, using the same applications and the same configurations as previously, were conducted to evaluate the performance of PVT-EA. In the developed model, we again used the VCI protocol specifications. The performance error with PVT-EA was reduced to zero for all the tested configurations. The reason behind this result is the using of in-order scaler processor. As our processor model is instruction accurate, interactions at the pipeline level are not taken into-account. Consequently, at this level the processor model could be replaced by a cycle accurate model,

0-7695-2975-5/07 $25.00

if an out-of-order superscalar processor is to be used. This point is proposed as a possible extension of the project. The speedup decreases by less than 30% compared to PVT-TA (Fig 10).

6.4 Modeling effort

Our approach also proved interesting in terms of modeling effort. It would allow designers to develop simulation platforms in less time. Using such a tool early in the design process reduces the exploration space, shortens the time-to-market, and increases the design team's productivity. Table 3 presents the modeling effort, in terms of lines of code (LOC), required to design an MPSoC platform for the three approaches: CABA, PVT-TA and PVT-EA. Given these results, the modeling effort using PVT-TA and PVT-EA is reduced, respectively, by a factor of 34.8% and 26.4%.

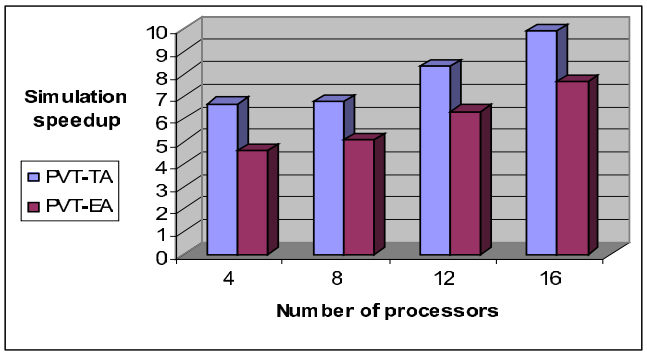


Figure 10. Speedup comparison

7 Conclusion

Modeling MPSoC architecture early in the design process decreases the developmental complexity of such systems and accelerates the design space exploration. In this paper, we propose a new PVT approach, with two sublevels namely PVT-TA and PVT-EA, intended to increase the speed of MPSoC simulation. Several component models have been developed using these two sublevels and enhanced with performance estimation tools to provide accurate execution time estimates. Simulation results demonstrate the complementarity between the PVT-TA and PVT-EA models. The first produces a higher simulation speedup factor and an average level of accuracy, while the second provides a more accurate estimate within a reasonable simulation time. Future research will focus on several areas. First, we plan to apply the same methodology to more complex architectures (VLIW and superscalar processors). Second, we hope to enhance simulation models with energy estimation tools for reliable design space exploration.

Table 3. Comparison of modeling effort

Abstract level	Modeling effort(LOC)					Reduction (%)
	Proc	Cache	Inter	Mem	Total	
CABA	1578	553	399	312	2842	
PVT-EA	1486	244	177	183	2090	26.46
PVT-TA	1259	238	176	180	1853	34.80

References

[1] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4), Apr. 2003.

[2] R. Ben Atitallah et al. Estimating energy consumption for an MPSoC architectural exploration. In *ARCS'06*, Frankfurt, Germany.

[3] L. Benini et al. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Springer J. of VLSI Signal Processing*, 2005.

[4] L. Benini et al. SystemC cosimulation and emulation of multiprocessor SoC designs. *IEEE Computer, vol. 36, no. 4*, April 2003.

[5] CACTI home page. `http://research.compaq.com`.

[6] L. Cai et al. Transaction level modeling: an overview. In *CODES+ISSS '03*, New York, USA.

[7] G. Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: Video Coding at Low Bit Rates. *IEEE Trans. On Circuits And Systems For Video Technology*, November 1998.

[8] A. Donlin. Transaction level: flows and use models. In *CODES+ISSS '04*, Stockholm, Sweden.

[9] F. Fummi et al. Native ISS-SystemC integration for the cosimulation of multi-processor SoC. In *Date'04*, Paris, France.

[10] D. Gajski et al. *SpecC:Specification Language and Methodology*. Kluwer, 2000.

[11] F. Ghenassia. *Tansaction level modeling with SystemC*. Springer, 2005.

[12] GRACE++. *System Level Design Environment for Network-on-Chip (NoC) and Multi-Processor platform (MP-SoC) exploration*.

[13] T. Groetker et al. *System Design with SystemC*. Kluwer, 2003.

[14] D. Kim, Y. Yi, and S. Ha. Trace-driven HW/SW cosimulation using virtual synchronization technique. In *Design Automation Conference'05*, Anaheim, California.

[15] R. Krishnan et al. Design of a 2D DCT/IDCT application specific VLIW processor supporting scaled and sub-sampled blocks. *16th Int. Conf. on VLSI Design*, 2003.

[16] H. Lee, B. Lee, Y. Lee, and B. Kang. Optimized VLSI design for enhanced image downscaler. In *2nd IEEE AP-ASIC 2000*, Cheju Island, Korea.

[17] C. Loeffler et al. Practical fast 1-D DCT algorithms with 11 multiplications. In *ICASSP'89*, Glasgow, UK.

[18] MIPS Technologies Consortium. `http://www.mips.com/`.

[19] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Conference on Languages, compilers and tools for embedded systems*, Berlin, Germany, 2002.

[20] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based on-chip communication architectures. In *CODES+ISSS'04*, Stockholm, Sweden.

[21] SoCLib project. 2003. `http://soclib.lip6.fr/`.

[22] E. Viaud, F. Pecheux, and A. Greiner. An efficient TLM/T modeling and simulation environment based on parallel discrete event principles. In *DATE'06*, Munich, Germany.

[23] Virtual Component Interface Standard. `http://www.vsi.org/`.

0-7695-2975-5/07 $25.00 © 2007 IEEE

An MDE Approach for Energy Consumption Estimation in MPSoC Design

Chiraz Trabelsi
INRIA Lille Nord Europe
France
chiraz.trabelsi@inria.fr

Samy Meftali
INRIA Lille Nord Europe
France
meftali@lifl.fr

Rabie Ben Atitallah
INRIA Lille Nord Europe
France
benatita@lifl.fr

Abderrazak Jemai
LIP2 Laboratory Faculty of Science
of Tunis, Tunisia
Abderrazak.Jemai@insat.rnu.tn

Jean Luc Dekeyser
INRIA Lille Nord Europe
France
dekeyser@lifl.fr

Smail Niar
INRIA Lille Nord Europe
France
niar@lifl.fr

ABSTRACT

Energy Consumption is a leading criterion to take into account in the design of multiprocessor systems on chip (MPSoC). In this paper, we present a solution to estimate the energy consumption early in MPSoC design in order to find a good performance/energy trade-off in the design flow. This solution is based on the injection of consumption estimators between the hardware components during the co-simulation of a system at the CABA (Cycle Accurate Bit Accurate) level. These estimators are designed using a design framework and the corresponding SystemC code is automatically generated thanks to a model driven approach. Our solution offers an energy estimation framework without changing the IP(Intellectual Property)source codes, using standalone estimation modules, which allows their reuse. The accuracy of this approach is checked by integrating the consumption estimation in the simulation of significant applications.

1. INTRODUCTION

The next generation of MPSoC are expected to accommodate multiple processors on the same chip, which increases the performance but increases at the same time the power consumption. As technology is moving quickly into the nano-meter domain, the energy dissipated by MPSoC is more and more significant. So, it becomes harder to guarantee sufficient battery life time. Thus, balancing performance and power becomes a major design challenge. Therefore, the ITRS roadmap lists power reduction as the second most important design challenge[3].

To crack the power problem while maintaining acceptable productivity requires power methodologies that support IP reuse and abstraction. They have also to be automatically integrated in simulation models. It is interesting also for power methodologies to be non-intrusive in the original system's simulation code. In fact, without this last constraint, it is hard to imagine using multi provider IP libraries that are absolutely necessary in modern systems design.

The objective of this work is to provide designers by a complete and systematic performance and power estimation methodology for MPSoC design. This methodology has three major characteristics:

- Automatic: to reach time-to-market constraints, the power consumption estimation methodology has to be automatic, fast and not tedious for users,
- Allowing reuse and not intrusive: the solution must support heterogeneous and multi providers IP reuse. It is also necessary to keep the original system's code "clean", so to be non-intrusive. This is particularly important because, generally, designers do not have source code of IPs provided by foreign companies.
- Easy integration in a complete design flow: to be efficient, a power consumption methodology must be easily used in complete and automatic design flows.

This paper is organized as follows: in the next section, we explain and classify the main approaches of the literature. Details of the proposed methodology are given in section 3. In section 4 a significant case study is presented and the simulation results are analyzed. Section 5 concludes this article and gives some of the perspectives of this work.

2. RELATED WORK

Although the accuracy of consumption estimation tools at low levels [4], the simulation requires a high amount of time. This limits represent an obstacle to apply this approach to complex systems. With the increasing complexity of designs, these estimation approaches become inadequate as they produce estimation results late in the design flow. More abstract estimation techniques are required to enable early design decisions. To achieve this goal, several studies have proposed evaluating power consumption at higher abstraction levels such as the CABA level, on which this work is based. At this level, the behavior of components is simulated cycle by cycle using an architectural level simulator. An analytic power model is used to estimate consumption for each platform component. The power model of a component is based on the power costs of pertinent activities and the occurrences of these activities during the simulation. The consumption of the main internal units is estimated using power macro-models, produced during lower-level simulations. The power model is integrated into the simulator. During the simulation, the consumption is calculated in every cycle, which permits accurate estimates. The contributions of the internal unit activities are calculated

and added together during the simulation. Among tools described with this approach, we find SimplePower[12] and MPARM[5]. The main drawback of such tools is that they use intrusive approaches. A solution for this problem is to use standalone estimators connected between components. Among tools using this approach, we cite UNISIM[6], which is a simulation framework that offers cycle accurate energy estimation. The estimation is based on shadow modules. A shadow module is connected to the module that is being monitored. Whenever an operation is correctly executed by the monitored module, the shadow module is notified. This approach has many similar points with ours: both use separate modules in order to estimate power consumption, both use the inputs of the monitored modules for power estimation and both are adaptable for many hardware configurations. The disadvantage of shadow approach is that it uses a specific communication protocol between hardware components and between these components and the estimation modules. Besides, in the UNISIM framework, the monitored modules need to notify the estimation modules whenever an operation has been performed. This means that the IP source codes have to be changed in order to support this functionality.

An interesting approach to reduce the design time of MPSoC is the Model Driven Engineering approach. MDE revolves around three focal concepts. Models, Metamodels and Model Transformations. The MDE development process starts from a high abstraction level and finishes at a targeted model, by flowing through intermediate levels of abstraction via Model Transformations (MTs) [9]; by which concrete results such as an executable model (or code) can be produced. MTs carry out refinements moving from high abstraction levels to low levels models and help to keep the different models synchronized. At each intermediate level, implementation details are added to the MTs. A MT is a compilation process that transforms a source model into a target model and allows to move from an abstract model to a more detailed model. Usually, the initial high level models contain only domain specific concepts, while technological concepts are introduced seamlessly in the intermediate levels. The source and target models each conform to their respective metamodels thus respecting exogenous transformations [7]. A model transformation is based on a set of rules (either declarative or imperative) that help to identify concepts in a source metamodel in order to create enriched concepts in the target metamodel.

The solution proposed in this paper makes a trade-off between the speed of simulation using a high level of abstraction and an acceptable accuracy compared with lower levels. Besides, it is non-intrusive, which is interesting especially if the source code of the IPs is not accessible. Furthermore, the consumption estimators are automatically generated using a Model Driven Engineering approach, which permits a gain in the design time.

3. CONSUMPTION ESTIMATION AT THE CABA LEVEL

3.1 The Non-Intrusive Approach

As we said in the related work section, at the CABA level, the total energy consumption of a system is obtained by adding the consumptions of system components together. To provide accurate estimation, two kinds of consumption

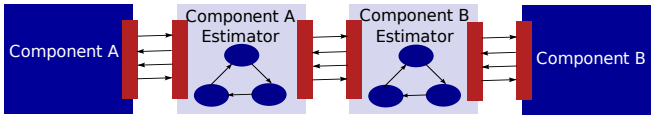


Figure 1: Consumption Estimators

are considered: dynamic consumption related to component activity (e.g. read/write operations), and static consumption related to leakage currents. For a long time, dynamic power consumption has been considered more significant then static consumption. However,this point of view changed with the advent of new sub-micron technologies, for which both types of consumption have their degree of importance. The energy consumption of a hardware component follows this equation:

$$E = \sum_i N_i * C_i$$

where N_i is the number of times the activity i is realized or the number of cycles the component is inactive, and C_i is the cost of a unit of the activity i or of a cycle of inactivity. In a previous work [1], we developed energy models for the main components of the SoCLib library: the processor, the cache memory, the shared SRAM memory and the interconnection network. We determined each component pertinent activities and we measured their unit costs using low level tools. To determine the occurrences of each activity, a counter was inserted in the source code of the components for each activity kind. Each counter is incremented during the simulation if the corresponding activity occurs during the current cycle. This approach gives the consumption of the whole architecture in every cycle. The values of the activity counters are transmitted to the energy consumption models integrated into the SoCLib simulator, to accumulate the energy dissipation of the architecture. The consumption simulator contains an energy model for each hardware component, which depends on its technology parameters.

This approach gives fast accurate results but has the inconvenience to modify the source code of IPs, which is supposed to be maintained clean. Besides, generally, designers do not have this code if the IPs are provided by foreign companies. So, how can we determine the occurrences of the activities without being intrusive? The solution is to detect these activities through the signals that the components exchange. For this reason, we connect a number of consumption estimation modules between the components in order to determine their activities from the requests and responses that they exchange. Figure 1 shows that to determine the consumption of two linked components, we need a consumption estimator for each one. We don't use only one estimator to estimate the consumption of both components because separating estimators allows their reuse with different architectures.

The components of SoCLib library have VCI[11] interfaces besides a specific protocol for the communication between processors and caches. We chose the OCP[8] protocol for the interfaces of our estimators. This protocol allows

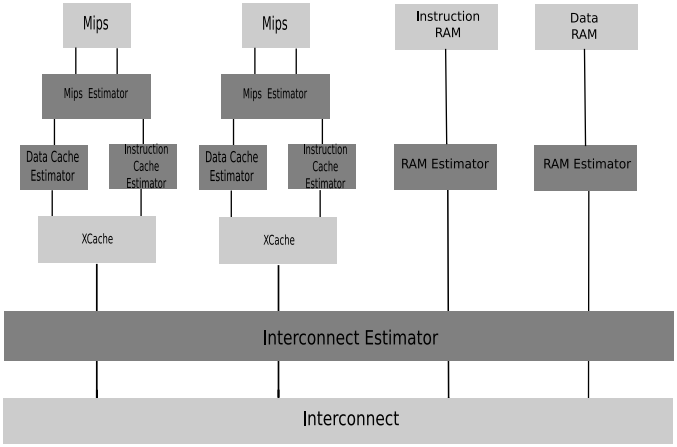


Figure 2: Example of simulated architectures with consumption estimators

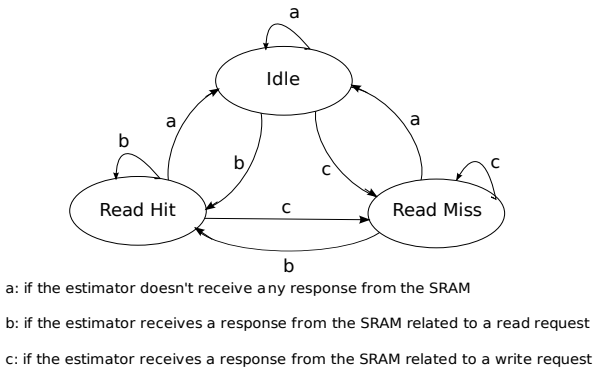


Figure 3: The SRAM Estimator's FSM

any type of communication and so it can be used for estimators connected between processors and caches as well as between the other VCI components. This solution permits the use of a single standard protocol allowing the integration of the estimators in different architectures. Therefore, we used wrappers to make the evaluated components compliant with the OCP protocol so we can connect estimators to them.

3.2 Consumption estimators for SoCLib

In this study, we developed consumption estimators for certain SoCLib components: the MIPSR3000, the cache memory, the SRAM shared memory and the interconnect. Figure 2 shows an example of a SoCLib architecture with integrated consumption estimators. The description of an estimator at the CABA level is based on a power state machine. The power state machine contains relevant power states of a component and transitions between them. A power state is related to an activity of the component or corresponds to a static consumption. The transitions are conditioned by the values of the signals that the monitored component exchange with others. Depending on these signals, it is possible to determine the current activity of a component. At every cycle, there is a transition in the power state machine. In each transition, there is an energy consumption related to the target state and so the corresponding occurrence counter is incremented.

For an SRAM memory, three main activities consume energy: Read, Write and Idle. These activities correspond to the read access mode, the write access mode and the waiting state. This approach is similar to the approach proposed by Loghi et al[5]. Thus, the SRAM Estimator's FSM is composed of three states: Read, write and Idle as shows Figure 3. Connected between the SRAM and the interconnect, the SRAM estimator intercepts the requests that the SRAM receives and the responses that it sends. If the SRAM estimator receives a response from the SRAM corresponding to a read request, there will be a transition to the Read state. If the SRAM doesn't send any response, that means that it's inactive, which corresponds to the Idle state in the FSM. So, the SRAM total energy follows this equation:

$$E_{SRAM} = n_{read}.E_{read} + n_{write}.E_{write} + n_{idle}.E_{idle}$$

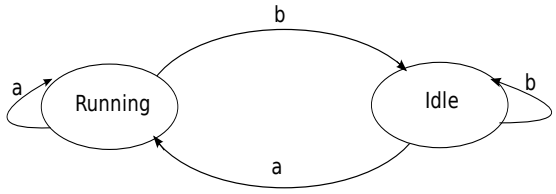
where n_{read}, n_{write} and n_{idle} are respectively the number of occurrences of a read access, a write access and a cycle of inactivity of the SRAM. These occurrences are given by the counters associated to each state in the FSM. E_{read}, E_{write} and E_{idle} are respectively the costs of a read access, a write access and a cycle of inactivity. These costs depend on the number of words and the number of bits per word in the SRAM and were determined using the ELDO simulator[1].

Figure 4 shows the FSM of an estimator of a MIPSR3000 processor. The FSM of this estimator has two states: a state of activity where the MIPS executes an instruction and a state of inactivity where it waits for cache responses or does nothing. This estimator is connected between the processor and the cache. The detection of the MIPS activities is based on the responses that it receives from the instruction and data caches. If the MIPS estimator receives a cache miss it deduces that the MIPS will be inactive because of this miss, so it increments the idle counter. This counter is incremented also when the MIPS estimator doesn't receive any response from the caches, in this case the processor is not executing any instruction. If the MIPS estimator receives a valid response from the cache, and both cache miss signals are set to false, it deduces that the MIPS is executing an instruction and increments the running counter.

The processor's energy consumption in the active state depends on the instruction to be executed. The set of all the instructions energies will constitute the processor energy model, and the cost of each instruction can be determined from low level measurements. This approach has been used by several other authors [5] [10]. For instance Sinha [10] demonstrated that for the StrongARM SA-1100 processor, the maximum current variation between instructions during the execution of a program is only 8%. The Hitachi HS-4 processor behaves in the same way [10]. Consequently, for a processor with a relatively simple architecture, like the MIPS R3000, a consumption model that considers only the average current per instruction is sufficient. Thus, we can consider an average running energy cost. So, the energy consumed by the MIPS follows this equation:

$$E_{MIPS} = n_{running}.E_{running} + n_{idle}.E_{idle}$$

To estimate the energy consumption of more complex processors, we have to take into account the consumption of



a: if the mips estimator receives a response from the data or the instruction cache and there is no cache miss in both caches

b: if the mips estimator receives a cache miss signal or there is no response from both caches

Figure 4: The MIPS Estimator's FSM

each instruction. The power state machines were also used to represent the consumption of the instruction and data cache, and the interconnect.

3.3 The MDE Approach

Our work aims to provide MPSoC designers with an environment that allows them to generate automatically consumption estimators from estimator models. To modify an estimator, they are not obliged to write long code lines, they only need to enter a few modifications in the estimator model and relaunch the generation process to obtain the desired result. Thus, Model Driven Engineering is a solution to make power consumption estimation a fast and not tedious work for users in order to respect the time-to-market constraints. The integration of the consumption estimation in the design flow of the Gaspard framework[2], which is based on MDE too, allows thus fast architectures exploration. To implement this integration, we used the same MDE tools as Gaspard, namely Papyrus for graphical modeling, QVTO for model transformations and JET for code generation.

To facilitate the modeling of the estimators for the users, we have designed a UML profile so they can have a graphical view of the estimator structure, FSM and deployment. Here we proposed a new profile because there are notions that we need and that we don't find in standard UML profiles such as MARTE that Gaspard uses to model MPSoC. In fact, what we need here is to model the internal structure of a component (the estimator) to be generated later, while MARTE permits to model architectures composed of components that already exist. Among the notions that we don't find in MARTE is the State Machine that is a main concept in our modelling.

The model elaborated using the proposed profile must be transformed to a target model compliant to the consumption estimation metamodel. The code of the estimators is then generated using the target model. Figure 5 shows the consumption estimation metamodel. It is based on an "EstimationModel" that may contain one or several "ConsumptionEstimators" , which allows to have more than one estimator in the same model. A "ConsumptionEstimator" has "Interfaces" which may be OCP interfaces or any other type of interfaces. Provided that the "ConsumptionEstimator" will be connected between hardware components, it has to transmit signals between them. This work is done by transmitting signals between each two associated interfaces, this information is given by the association "isAssociatedTo". The interfaces have "InterfaceTypes" such as an OCP master or an OCP slave. To each "InterfaceType" are associated input and output ports. To take into account the configuration of the monitored component, the estimator uses some "Parameters" ("configurationParameters") which may also serve in the energy consumption estimation ("consumptionParameters") such as the numbers of words and lines in the cache. In certain cases, the estimator has to save some data in variables in order to use them as a condition for next transitions. For example, the cache estimator needs to know if there was a cache miss previously, that's the use of the association "otherParameters". Saving the values of these parameters is an "activity" which accompanies a transition or is a "doActivity" of a state in the estimator FSM. A "Parameter" has a "DataType" and a "multiplicity" in order to indicate if it's an array. The size of the array is also a "Parameter" of the "ConsumptionEstimator". The path to the "InterfaceType" source code is given by the "CodeFile" metaclass. The FSM of the estimator is inspired from the state machine of the UML metamodel. An estimator has a "StateMachine" which contains "States" and "Transitions" between "States". A "State" may have a "doActivity". A "Transition" may have a "condition" and an "effect". An "Activity" manipulates several "EstimatorElements". The source code path of an activity is given by the "CodeFile" metaclass . That supposes that all the activities of an estimator are saved in the same file with the function that calculates the consumption of the monitored component and whose name is given by the property "estimationFunctionName" of the "ConsumptionEstimator" metaclass.

Thanks to the profile, the MPSoC designer can elaborate easily the estimator models. Figure 6 shows the state machine diagram corresponding to the instruction cache estimator's FSM. Here, we have omitted to display the guards and effects of transactions on the connectors because they are long expressions and would make the figure illegible. This information can be viewed in the properties view when clicking on the connectors. Figure 6 shows the guard and effect of a transition between the IDLE state and the Read Miss state(RUPDT), which corresponds to the update of the cache content after a read miss.

After the Model To Model transformation. The target model is transformed into a SystemC code. With this process, we have generated estimators with 200 to 300 code lines compared with 10 to 120 lines inserted into IPs with the intrusive approach. This difference is because of the code necessary to manage the FSM of the estimator and especially the conditions of the transitions. Here, MDE approach saves us a long coding time.

4. SIMULATION RESULTS

Gaspard is an environment that permits to model a whole MPSoC architecture using UML and then generate the simulation code in various abstraction level languages such as VHDL and SystemC TLM[2]. For the moment, the code generation of a whole architecture at the CABA level is not supported yet but it is not very different from the code generation at the TLM level. The integration of the consumption estimators in the architecture in a UML model either in CABA or TLM levels can be done by indicating that a component will be monitored by an estimator in the deployment diagram. For Example, the deployment diagram can

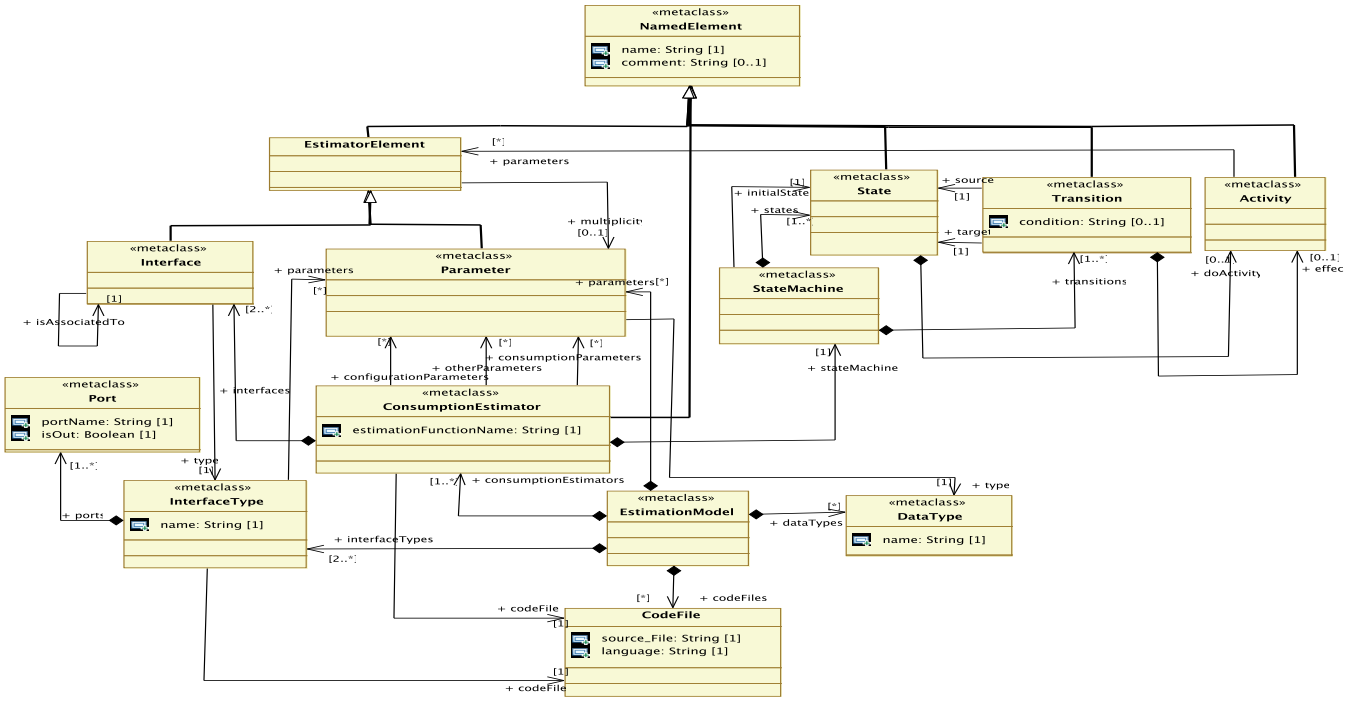


Figure 5: Consumption Estimation Metamodel

indicate that the RAM memory that is used is OCP compliant, and that its estimator is connected to its OCP slave interface. So, at the generation of the code for the simulation, there will be an insertion of an estimator between the memory and the other component of the MPSoC to which the memory is normally connected. That is what we will implement in future works. For now, to validate our approach we have to use it in an architecture described at the CABA level. Provided that the connections between the components and the estimators cannot be done yet at the CABA level using Model Driven Engineering, we used instead some script shells that generate different architectures giving as parameters the number of processors, the cache size, etc. So, the previously developed energy models were integrated into the SoCLib architectural simulator in order to profit from a fast architectural exploration environment for MPSoC design. To validate our approach, we used a Visiophony application for the UMTS network. For this application, we chose a minimal resolution using the QCIF format (144*176 pixels). The coding standard chosen was the H.263, adapted for Visiophony and Videoconference applications. For this paper, we evaluated only the DCT task to validate our approach. Figure 2 shows an example of the architectures used in the simulations. The hardware components used in these architectures are MIPSR3000 processors, 16KB SRAM memories and micro-networks. The used caches contain actually two independent instruction and data caches, sharing the same interface. They are direct mapped caches. The data cache's writing policy is write-through. To prove the accuracy of our approach, we have to compare its consumption results with those of the intrusive approach followed in our previous work [1]. Therefore, we executed the DCT task on different architectures varying the number of processors and the cache size, and we did the same simulations using the previous and the new approaches for consumption estimation.

The results of the simulations showed that the non-intrusive approach gave estimation values which are very close from those of the intrusive one(the consumption results of the previous approach can be found in[1]) with a difference that does not exceed 0.3% as shows Figure 7. The difference between the results given by the two approaches shows that there are some internal details that we cannot detect only through the interfaces. In fact, in this case, the difference is due to the fact that the consumption of the request FIFO queue of the cache is not considered in the new approach but in the previous one. This consumption corresponds to the consumption of a memory which is the FIFO here. As we have seen, to determine the consumption of a memory, we have to determine the number of the accesses to that memory and its size. The number of the requests written into the FIFO can be given by the read misses, the uncacheable reads and the write counters of the cache estimator, but the information that we don't have here is the maximum size of the FIFO. This can be only known by accessing the source code of the cache IP, which is against the principles of the non-intrusive approach. However, provided that the consumption of the FIFO is very small, it can be neglected, and we can say that using standalone estimation modules permitted to get accurate results because it took into account the main pertinent and the most consuming activities of the monitored hardware components.

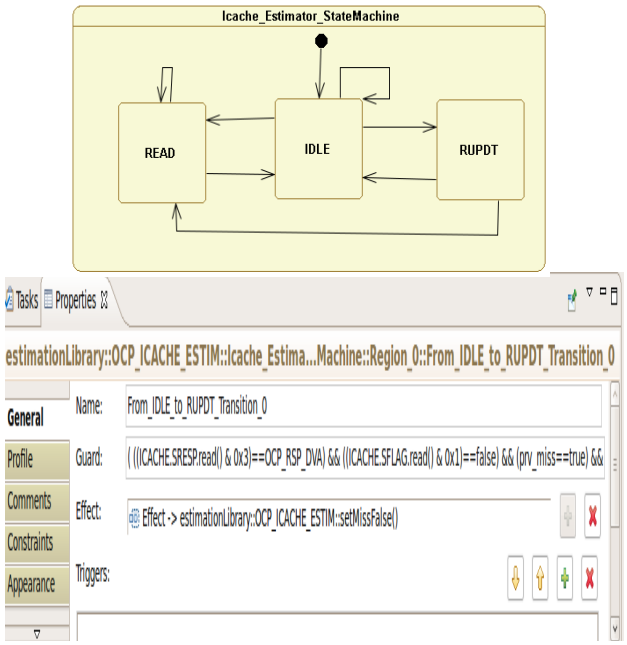


Figure 6: MIPS Estimator's FSM model

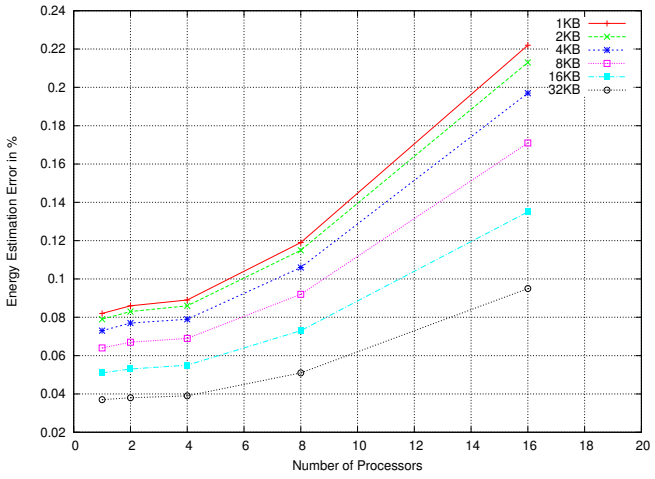


Figure 7: Variation in energy consumption estimation error in terms of cache size and number of processors

5. CONCLUSION

In this study, we enhanced our previous work in energy estimation, which used an approach that is intrusive in IP code sources in order to detect the activities of a hardware component and determine their consumption costs. In this paper, we have presented a non-intrusive approach using standalone estimators for energy consumption. These modules can be connected between hardware components in order to estimate their consumption. The implementation and the test of this approach followed these steps: designing consumption estimators at the CABA level using the Gaspard framework, generating automatically these estimators thanks to MDE tools, and integrating them in the SoCLib simulator to validate their estimation results. The components studied here are simple and their activities are fully detectable through the signals that they exchange. We plan to adapt the same approach for more complex architectures in order to obtain an acceptable accuracy and to validate our approach with different applications. Besides, we intent to apply the same approach for higher abstraction levels such as TLM.

6. REFERENCES

[1] R. B. Atitallah, S. Niar, A. Greiner, S. Meftali, and J. Dekeyser. Estimating energy consumption for an mpsoc architectural exploration. In *ARCS*, Frankfurt, Germany, 2006.

[2] Gamatié and al. A model driven design framework for high performance embedded systems. Technical report, INRIA, 2008.

[3] ITRS. Itrs 2007 edition. http://www.itrs.net/.

[4] R. P. Llopis and K. Goossens. The petrol approach to high-level power estimation. In *International Symposium on Low Power Electronics and Design*, USA, 1998.

[5] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *GLSVLSI*, Boston, Massachusetts, USA, 2004.

[6] D. Ludovici, G. Keramidas, G. N. Gaydadjiev, and S. Kaxiras. Integration of power saving techniques in the unisim simulation framework through the shadow module design paradigm. In *Rapid Simulation and Performance Evaluation*, 2009.

[7] T. Mens and P. V. Gorp. P. a taxonomy of model transformation. *Proceedings of the International Workshop on Graph and Model Transformation, GraMoT 2005*, 152:125–142, 2006.

[8] OCP. *OCP International Partnership.* http://www.ocpip.org/home.

[9] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.

[10] A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Proceedings of the 38th DAC Conference*, 2000.

[11] VCI. *Legacy Documents of the VSI Alliance.* http://vsi.org/.

[12] W. Ye, N. Vijaykrishnan, M. Kandemir, , and M. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *DAC*, Los Angelos, California, 2000.

Targeting Reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation

Imran Rafiq Quadri, Abdoulaye Gamatié, Samy Meftali, Jean-Luc Dekeyser

INRIA Lille Nord Europe - LIFL - USTL - CNRS, FRANCE
E-mail: {firstname.lastname}@lifl.fr

Huafeng Yu*

IRISA/INRIA Rennes - Bretagne Atlantique, FRANCE
E-mail: Huafeng.Yu@inria.fr
* Corresponding author

Éric Rutten

INRIA Grenoble Rhône-Alpes, FRANCE
E-mail: Eric.Rutten@inria.fr

Abstract:
As SoC design complexity is escalating to new heights, there is a critical need to find adequate approaches and tools for handling SoC co-design aspects. Additionally, modern reconfigurable SoCs offer advantages over classical SoCs as they integrate adaptivity features to cope with mutable design requirements and environment needs. This paper presents a novel approach for addressing system adaptivity and reconfigurability. A generic model of reactive control is presented in a SoC co-design framework: Gaspard2. Afterwards, control integration at different levels of the framework is illustrated for both functional specification and FPGA synthesis. The presented works are based on Model-Driven Engineering and the UML MARTE profile proposed by Object Management Group, for modeling and analysis of real-time embedded systems. Our contributions thus relate to presenting a complete design flow to move from high level MARTE models to automatic code generation, for implementation of dynamically reconfigurable SoCs.

Keywords: Intensive Signal Processing; UML; MARTE; Model-Driven Engineering; SoC; Reconfigurability; FPGAs.

Biographical notes: Imran Rafiq Quadri is currently finishing his PhD thesis in Computer Science from the University of Lille 1. He is equally also a research and teaching assistant at University of Lille 1. He achieved his Master's degree from University of Lille 1 in the field of embedded systems in 2006, and is currently involved in the DaRT project at INRIA Lille-Nord Europe; working in the field of partial dynamic reconfiguration and FPGA synthesis.

Abdoulaye Gamatié received a PhD degree in Computer Science in 2004 from University of Rennes 1, where he also held an assistant professor position for two years. In 2005, he was a postdoctoral fellow at INRIA Futurs. Since 2006, he is a research scientist at CNRS, in France. His research interests include methodologies and tools for the reliable design of embedded systems in general.

Samy Meftali obtained his PhD in Computer Science from University of Grenoble 1 in 2001. Since then, he is an associate professor at University of Lille 1 in France. His research interests are mainly modeling, simulation and FPGA implementation of intensive signal processing embedded systems.

Jean-Luc Dekeyser received his PhD degree in computer science from the University of Lille 1 in 1986. After a few years at the Supercomputing Computation Research Institute in Florida State University, he joined University of Lille 1 as an assistant professor. He is currently a professor at University of Lille 1 and is also heading the DaRT project at INRIA Lille-Nord Europe.

Huafeng Yu is currently an expert research engineer at IRISA/INRIA Rennes, France. He received his PhD degree in Computer Science from University of Lille 1 in France in 2008. His research interests concentrate on safe design of embedded systems through formal methods.

Éric Rutten, PhD 1990 and Hab. 1999 at University of Rennes, France, works at INRIA in Grenoble, in the field of reactive embedded systems. He is currently working on model-based control of adaptive and reconfigurable computing systems, using control techniques; particularly discrete controller synthesis.

1 Introduction

Since the early 2000s, Systems-on-Chip or SoC has emerged as a new methodology for embedded systems design. In a SoC, the computing units, e.g., programmable processors, memories and I/O devices, are all integrated into a single chip. These SoCs are generally dedicated to applications like multimedia video codecs, software-defined radio and radar/sonar detection systems, that require intensive data-parallel computations. Unlike general parallel applications that focus on code parallelization, data-parallel applications concentrate on regular data partitioning, distribution and their access to data.

1.1 Motivations

As the computational power increases for SoCs in accordance with Moore's law (Moore 1965), more functionalities are expected to be integrated into the system. As a result, more complex software applications and hardware architectures are integrated, leading to a *system complexity* issue which is one of the main hurdles faced by SoC designers. The consequence of this complexity is that the system design, particularly software design, does not evolve at the same pace as that of hardware. This has become a critical issue and has finally led to the *design productivity gap*.

Adaptivity and *reconfigurability* are also critical issues for SoCs which must be able to cope with end user environment and requirements. For instance, mode-based control plays an important role in modern embedded systems by permitting description of Quality-of-Service or QoS choices: 1) changes in executing functionalities, e.g., color or black and white picture modes for modern digital cameras; 2) changes due to resource constraints of targeted hardware/platforms, for instance switching from a high memory consumption mode to a smaller one; or 3) changes due to other environmental criteria such as communication quality and energy consumption. A suitable control model must be generic enough to be applied to both software and hardware design aspects.

For implementing dynamically reconfigurable SoCs, *Field Programmable Gate Arrays* or FPGAs are considered as an ideal solution, due to their inherent reconfigurable nature. Designers can initially implement, and afterwards, reconfigure a complete FPGA based SoC for the required customized solution. Thus FPGAs offer a migration path for final *Application Specific Integrated Circuit* or ASIC implementation. State of the art FPGAs can also change their functionality at *runtime*, known as *Partial Dynamic Reconfiguration* or PDR (Lysaght et al. 2006). This feature allows to modify specific regions of an FPGA on the fly, with the advantage of time-sharing the available hardware resources for executing multiple mutually exclusive tasks. It permits context switching depending upon application needs, hardware limitations and QoS requirements. Currently only Xilinx FPGAs fully integrate partial dynamic reconfiguration. These FPGAs also support internal self dynamic reconfiguration, in which an internal controller, e.g., a *hardcore*/*softcore* embedded processor, manages the reconfiguration aspects.

1.2 Elevation of design abstraction levels

An effective solution to SoC co-design problem consists of raising design abstraction levels. The important challenge is to find efficient design methodologies that raise design abstraction levels to reduce overall complexity. These methods must effectively handle issues like accurate expression of inherent system parallelism, such as application loops and hierarchy. They should also be able to express the control at higher abstraction levels to integrate adaptivity and reconfigurability features in modern embedded systems.

Unified design approach is an emerging research topic for addressing the various issues related to SoC co-design. High level SoC co-modeling design approaches have been developed such as Model-Driven Engineering or MDE (OMG 2007*b*). MDE enables high level system modeling of both software and hardware, with the possibility of integrating heterogeneous components into the system. *Model transformations* (S. Sendall and W. Kozaczynski 2003) can then be carried out to generate executable models from the high level models. MDE is supported by several standards and tools.

Gaspard2 (INRIA DaRT team 2009, Gamatié et al. 2010) is an MDE-based SoC co-design framework dedicated to specification of parallel hardware and software. It is based on the standard UML MARTE profile (OMG 2008); and allows to move from high level MARTE models to different execution platforms. It exploits the inherent *parallelism* included in repetitive constructions of hardware elements or regular constructions, such as application loops. The applications targeted by Gaspard2 also focus on a specific application domain, that of intensive data-parallel computations applications.

1.3 Our contribution

In this paper, we present a generic control semantics for the specification of system adaptivity and specially dynamic reconfigurability in SoCs. The introduced control semantics are integrated in Gaspard2 and are specified at a high abstraction level. This control semantics can be integrated at different SoC design levels, with an example being of the application level. However, for integrating aspects of dynamic reconfigurability, we propose integration at a design level that links the basic building blocks of applications/architectures to their *Intellectual Properties* or IPs. Integration at the IP deployment level focuses on FPGA synthesis and is specially oriented towards partial dynamic reconfiguration. Our design flow generates two key concepts related to a dynamically reconfigurable FPGA based SoC. Firstly, we generate the code for a dynamically reconfigurable region, which relates to a high level application model, translated into an hardware functionality, e.g., an hardware accelerator and its different implementations, by means of model transformations.

Secondly, the control semantics are utilized for the generation of the source code related to a reconfiguration controller, that manages the different implementations related to the hardware accelerator. Thus our design flow is mainly application-driven in nature.

Finally a case study related to a dynamically reconfigurable correlation module application is presented in the context of an anti-collision radar detection system, to validate our design methodology.

The rest of this paper is organized as follows. Related works are detailed in Section 2. An overview of the MDE-based Gaspard2 framework is provided in Section 3. Section 4 describes the control model in software applications, while Section 5 presents the control model for IP deployment and FPGA. Section 6 presents our case study. Control models used at different levels are compared in Section 7. Finally Section 8 gives the conclusion of the paper.

2 Related works

In this section, we detail some works in the domain of dynamically reconfigurable SoCs. We particularly focus on fine grain reconfigurable FPGA based SoCs, as compared to coarse grain reconfigurable architectures, of which numerous examples exist both in academic research and industry. Works related to reconfigurable SoCs can be categorized in several families: some works try to elevate design abstraction levels, such as providing specifications in system level languages like SystemC[1]; for decreasing the complexity related to creation of dynamically reconfigurable systems. Others deal with optimization directly at the *Register Transfer Level* or RTL by introducing new tools and methodologies.

2.1 Elevation of design abstraction levels

The MoPCoM project (Koudri et al. 2008) aims to target modeling and code generation of dynamically reconfigurable embedded systems using the UML MARTE profile for SoC co-design (Vidal et al. 2009). However, the targeted applications are relatively simple unlike those considered in the SoC industry. While the authors claim that they are capable of creating a complete SoC co-design framework, in reality, the high level application model is converted into a equivalent hardware design, with each application task transformed into an hardware accelerator in a target FPGA. Additionally, while the project permits modeling of the targeted FPGA architecture at the UML level as inspired from the works presented in (Quadri et al. 2009*b*,*d*), they are only capable of generating the *microprocessor hardware specification* file for input in Xilinx EDK tool, for manual manipulation of the partial dynamic reconfiguration flow. Moreover, IP reuse is not possible with this methodology.

In the OverSoC project (Pillement & Chillet 2009), the authors also provide a high level modeling methodology for implementing dynamically reconfigurable architectures. They integrate an operating system or OS, for providing and handling the reconfiguration mechanism. The global platform is conceptually divided into *active* and *reactive* components representing the reconfigurable architecture (an FPGA) and the OS respectively. The OS is executed on a general purpose processor interfacing with the FPGA. The active component is further composed of several sub components that represent the computation and reconfiguration components. The former relates to FPGA resources such as CLBs and LUTs, while the latter corresponds to an FPGA internal hardware reconfiguration core reponsible for the actual switching. Finally, SystemC is used for simulation and verification of the OS for managing the reconfigurable aspects. However, final implementation on FPGAs has not been carried out. It is up to the OS to determine whether an application task should be executed on the general purpose processor or the FPGA, depending upon the required resources.

A more complex OS is presented in (Bergmann et al. 2003), as an embedded uCLinux is used for managing partial dynamic reconfiguration. A customized device driver has been created to manage the hardware reconfigurable core, allowing users to carry out dynamic configuration in traditional Linux shell programs. However, the bitstreams are generated manually using the FPGA editor tool, raising chances of design errors.

(Brito et al. 2007) use a SystemC based design flow for implementing partial dynamic reconfiguration. The SystemC kernel is modified for the integration of reconfiguration operations for activation/disactivation of reconfigurable modules. Initial simulation is carried out using a SystemC model, which is then converted into a *Hardware Description Language* or HDL RTL model for actual implementation and comparison. The drawback of this approach is that the reconfiguration time related to module is predetermined by the designers. Additionally, the system only provides *on-off* functionality for the modules resulting in a simplified design with respect to partial dynamic reconfiguration.

In (Nezami et al. 2008), HandleC is used to implement partial dynamic reconfiguration for Software defined Radio, however, they only provide the design methodology and no actual implementation is carried out. In (F. Berthelot and F. Nouvel and D. Houzet 2008), a SynDEx based design flow is presented to manage SoC reconfigurability via implementation in FPGAs, with the application and architecture parts modeled as components.

2.2 Targeting coarse grain architectures

There also exists a large number of research works like Chameleon (Salefski & Caglar 2001), Rapid (Ebeling et al. 1996) and projects such as Morpheus (Morpheus 2010) that deal with coarse grain reconfigurable architectures. However these projects and works normally consider a lower abstraction level.

SPEEDS! or *Speculative and Exploratory Design in Systems Engineering* (SPEEDS! 2009) is also an European project for embedded systems development based on SysML and AUTOSAR. Equally, the EPICURE project (J.P. Diguet and G. Gogniat and J.-L. Philippe et al

2006) defines a design methodology in order to bridge the gap between high abstract specifications and heterogeneous reconfigurable architectures. The framework is based on Esterel design technologies and provides verification and synthesis capabilities. However, one of the existing drawbacks of this framework is the lack of available support for a high abstraction level design methodology, in order to reduce design complexity. Additionally, works such as Molen (Panainte et al. 2007) propose efficient compilers for reconfigurable architectures. The proposed Molen compiler was implemented on a Virtex II FPGA, and takes into account details such as related to synthesis placement conflicts as well as reconfiguration latencies.

2.3 High level control methodologies

In (Latella et al. 1999, Schäfer et al. 2001), the authors concentrate on control based modeling and verification of real-time embedded systems in which the control is specified at a high abstraction level via UML state machines and collaborations; by using model checking. A similar approach has been presented in (Faugere et al. 2007). However, control methodologies vary in nature as they can be expressed via different forms such as Petri Nets (Nascimento et al. 2004), or other formalisms such as mode automata (Maraninchi & Rémond 2003).

Mode automata extend synchronous dataflow languages with an imperative style, but without many modifications of language style and structure. They are a simplified version of Statecharts (Harel 1987) in syntax, which have been well adopted for the specification of control oriented reactive systems. Mode automata have a clear and precise semantics, which makes the inference of system behavior possible, and are supported by formal verification tools.

2.4 Methodologies of partial dynamic reconfiguration

For implementing partial dynamic reconfiguration in modern FPGAs, Xilinx initially proposed several design flows, which were not very effective leading to new alternatives. (Sedcole et al. 2005) presented an effective modular approach for 2-dimensional reconfigurable modules. Similarly, (Becker et al. 2003) implemented 1-dimensional modular reconfiguration using a horizontal slice based bus macro to connect the static and partially dynamic regions. They enhanced their works by placing arbitrary 2-dimensional rectangular shaped modules using routing primitives (Schuck et al. 2008).

In 2006, Xilinx introduced the *Early Access Partial Reconfiguration Design Flow* (Xilinx 2006) that integrated concepts introduced earlier in works such as (Sedcole et al. 2005) and (Becker et al. 2003). Researches such as (Bayar & Yurdakul 2008) and (Paulsson et al. 2007) focus on implementing softcore internal configuration cores on Xilinx FPGAs such as Spartan-3, that do not have the hardware internal reconfiguration cores, for effective implementation of PDR. Finally in (Koch et al. 2006), this reconfigurable core is connected with Network-on-Chip based FPGAs.

In comparison to the above related works, our proposition takes into account the following domains: SoC co-design, data intensive parallel computation applications, control/data flow, MDE, the UML MARTE profile, SoC adaptivity and PDR for fine grain reconfigurable FPGAs; which is the novelty of our design framework.

3 Gaspard2: a SoC co-design framework

Gaspard2 (INRIA DaRT team 2009, Gamatié et al. 2010) is an MDE oriented MARTE compliant SoC design framework as shown in Figure 1, providing an *Integrated Development Environment* or IDE dedicated to the visual co-design of embedded systems. The framework enables fast design and code generation with the help of UML graphical tools and technologies, such as Papyrus[2] and Eclipse Modeling Framework[3].

3.1 Repetitive model of computation

The Gaspard2 framework is based on a repetitive model of computation or MoC that relyies on the Array-OL specification language (Boulet 2007, 2008). The MoC describes the *potential parallelism* present in a system; and describes repetitive data intensive multidimensional computations; with the help of repetitive data dependencies. The manipulated data are in the form of multidimensional arrays, which have at most one possible *infinite dimension*. The arrays can be specified with a certain type specification, such as an array *shape*. The spatial and temporal dimensions are treated in the same manner, in the form of arrays. Particularly, time is expanded as one dimension of arrays. Additionally, access to data is carried out in the form of sub-arrays called *patterns*. In turn, in Gaspard2, data are also manipulated in the form of multidimensional arrays.

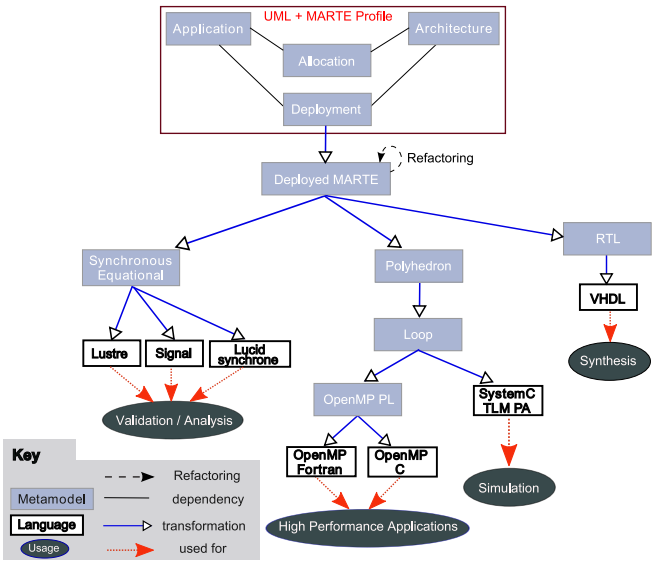


Figure 1: A global view of the Gaspard2 framework

3.2 Basic repetitive modeling concepts

Gaspard2 has also contributed to the conception of the UML MARTE profile. One of the key MARTE packages, the *Repetitive Structure Modeling* or RSM package is inspired from Gaspard2 and its model of computation. Additionally, some other packages such as the *Allocation* package and *Hardware Resource Modeling* package have also benefited from existing Gaspard2 concepts.

RSM enables the possibility to specify the shape of a repetition, by a multidimensionality, and also permits to represent a collection of potential links such as a multidimensional array. This repetition can be specified for an instance or a port of a component. The advantage is double fold: For hardware modeling, RSM presents a clear mechanism for expressing the links in a topology, as well as increasing the expression power of the mechanism for describing these complex topologies (Quadri et al. 2008). Complex regular, repetitive structures such as cubes and grids can be modeled easily via RSM, in a compact manner. Similarly for application aspects, RSM helps to determine different types of parallelism such as task and data parallelism.

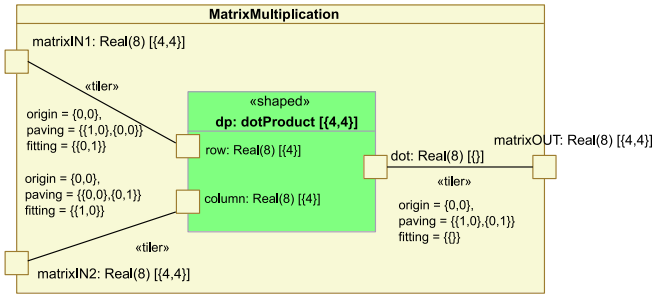


Figure 2: Representing *Data Parallelism* in Gaspard2 with the MARTE profile: The repetitive `MatrixMultiplication` component is composed of a repeated task: the `dP` instace of a `dotProduct` component. This repeated task represents the computing task, which takes one row and column from two input matrices; and produces one element in the final produced matrix. This task is elementary in nature and is thus represented differently from other tasks; and can be henceforth deployed, as discussed in section 5.1

A `repetitive component` such as shown in Figure 2 expresses the data parallelism in an application: in the form of sets of input and output `patterns` consumed and produced by the repetitions of the interior `part`. On the other hand, a `hierarchical` component such as illustrated in Figure 12 contains several `parts`; and defines a complex functionality in a modular manner. This concept thus provides a structural aspect of the application: specifically, task parallelism can be described using such a component.

The shape of a pattern is described according to a `tiler` connector which describes the tiling of produced and consumed arrays. The `reshape` connector allows to represent complex link topologies in which the elements of a multidimensional array are redistributed in another array.

An `interRepetition` dependency is used to specify an acyclic dependency among the repetitions of the same component, compared to a tiler, that describes the dependency between the repeated component and its owner component. Particularly, this dependency specification leads to the sequential execution of repetitions of the repeated part. A `defaultLink` connector provides a default value for the part repetitions that are linked with an `interRepetition` dependency, with the condition that the source of the dependency is absent. These last two RSM concepts have been illustrated in Figure 15 discussed later on in the paper, and are essential for control modeling.

3.3 Model transformations

Models in MDE are not only used for communication and comprehension but using model transformations (S. Sendall and W. Kozaczynski 2003), produce concrete results such as executable source code. With the help of metamodel(s) that define the concepts of their respective models, and to which these models conform to; models can be recognized by machines. As a result, they can be processed, i.e., a model is taken as input/source and then some models/targets are generated. This process is called a model transformation.

For the purpose of automatic code generation from high level models, Gaspard2 adopts MDE model transformations towards different execution platforms, such as targeted towards synchronous domain for validation and analysis purposes (Gamatié et al. 2008*b*, Yu et al. 2008, Yu 2008); or FPGA synthesis (Le Beux 2007, Quadri et al. 2009*b*), as shown in Figure 1. Model transformation chains permit moving from high abstraction levels to low enriched levels. Usually, the initial high level models contain only domain-specific concepts, while technological concepts are introduced seamlessly in the intermediate levels, by means of intermediate metamodels and respective models.

There exists a large number of transformation languages and tools such as ATLAS Transformation Language (INRIA Atlas Project n.d.), Kermeta (INRIA Triskell Project n.d.) among others; that support the *Meta-Object Facility Query/View/Transformation* or MOF QVT standard (OMG 2005) for model query and transformations. However, few of the QVT transformation tools are capable to execute large complex transformations such as present in the Gaspard2 framework. Also none of these engines is fully compliant with the QVT standard. However, some new tools such as QVTO (OMG 2007*a*) have recently emerged that implement the QVT Operational language, and are effective for handling the complex Gaspard2 model transformations.

For model transformations, Gaspard2 adapts QVTO as the de-facto transformation tool. It should be made evident that current model transformations are only uni-directional in nature. Similarly, Gaspard2 has adopted Acceleo[4], a code generation tool that is compliant with the MOF2Text standard[5].

Finally, Figure 3 shows the global overview of the model transformation chain related to implementing partial dynamic reconfiguration as dicussed in this paper. Initially a Gaspard2 application is modeled and deployed, along with

the associated control aspects in the Gaspard2 environment with Papyrus modeling tool; conforming to an extended version of the UML MARTE profile. This modeling is independent from any implementation details until the deployment phase.

Afterwards two *model-to-model transformations*, namely the *UML2MARTE* and *MARTE2RTL* transformations help to create an intermediate model, corresponding to its own metamodel, with concepts nearly equivalent to the *Register Transfer level* or RTL. This model is considered as a low abstraction level with details nearly corresponding to electronic RTL. The model provides details related to the hardware accelerators and the control features which can be used for eventual code generation. Finally using a *model-to-text transformation*, we generate the code related to different implementations of a hardware accelerator and the state machine for the reconfiguration controller. These aspects are detailed later on in the paper. As the model transformation rules are not trivial in nature and are about the size of several thousand lines of code, it is not possible here to give a generalized summary of the transformation rules present in our design flow.

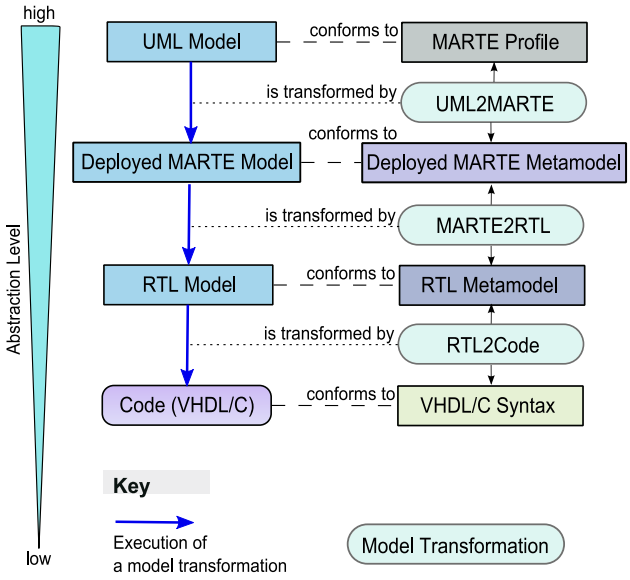


Figure 3: An abstract overview of the model transformation chain. Several intermediate metamodels help to bridge the gap between high level modeled UML diagrams and the final RTL code generation

3.4 Reactive control modeling semantics

This section provides the initial hypothesis related to the generic control semantics for expressing system reconfigurability. Several basic control concepts, such as `Mode Switch Component` and `State Graphs` are presented first. Then a basic composition of these concepts, which helps in eventual building of the mode automata, is discussed. This modeling derives from mode conception in mode automata. The notion of exclusion among modes helps to separate different computations. As a result, programs are well structured and fault risk is reduced.

3.4.1 Modes and Mode switch component

A *mode* is a distinct method of operation that produces different results depending upon the user inputs. A `Mode Switch Component` in Gaspard2 contains at least one mode; and offers a switch functionality that chooses execution of one mode, among several alternative present modes (Labbani et al. 2005). The mode switch component in Figure 4 illustrates such a component having a *window* with multiple tabs and interfaces. For instance, it has a mode value input port *m*, as well as several data input and output ports, i.e., i_d and o_d respectively. The switch between the different modes is carried out according to the *mode value* received through *m*.

The modes, M_1, ..., M_n, in the mode switch component are identified by the mode values: m_1, ..., m_n. Each mode can be hierarchical, repetitive or elementary in nature; and transforms the input data i_d into the output data o_d. All modes have the same interface (i.e. i_d and o_d ports). All the input and outputs share the same time dimension, ensuring correct one-on-one correspondence between the inputs/outputs. The activation of a mode relies on the reception of mode value m_k by the mode switch component through *m*. For any received mode value m_k, the mode runs exclusively. It should be noted that only mode value ports, i.e., *m*; are compulsory for creation of a mode switch component, as illustrated in Figure 4. Other type of ports, such as input/output ports are not always necessary and are thus represented with dashed lines.

3.4.2 State graphs

A state graph in Gaspard2 is similar to Statecharts (Harel 1987), which are used to model the system behavior using a state-based approach. We term these state graphs as *Gaspard state graphs*. A state graph can be expressed as a graphical representation of transition functions as discussed in (Gamatié et al. 2008*a*). A state graph is composed of a set of vertices called *states*. A state connects with other states through directed edges which are called *transitions*. Transitions can be conditioned by some *events* or Boolean expressions. A special label *all*, on a transition outgoing from state *s*, indicates any other events that do not satisfy the conditions on other outgoing transitions from *s*. Each state is associated with some mode value specifications that provide mode values for the state. Formal definitions of Gaspard state graphs have been presented in (Yu 2008). State graphs, can also be either parallely composed, or composed in an hierarchy; as illustrated in (Yu 2008).

The main difference between our state graphs and Harel Statecharts is that the former are transition functions in which neither initial or final states are defined. This is not the case in StateCharts or more generally in automata. The way transitions are fired in our state graphs and the information related to current state of the state graph is determined by using `interRepetition` dependencies. Details related to this aspect can be found in section 3.5.

A state graph in Gaspard2 is associated with a `Gaspard State Graph Component`, as shown in Figure 4. Thus a state graph determines the internal behavior of a Gaspard state graph component. A Gaspard state graph component determines the mode value definition by means of its associated state graph. The mode values allow to activate different exclusive computations or modes in the related mode switch components. Thus, Gaspard state graph components are ideal complements of mode switch components, with mode values being the relation between the two concepts. A Gaspard state graph component can be viewed as a controller component, while the mode switch component switches between the modes according to the present controller.

Similarly to the mode switch component, a Gaspard state graph component has its interfaces. These interfaces include event inputs from the environment, source state inputs, target state outputs and mode outputs. Event inputs are used to trigger transitions present in the associated Gaspard state graph. The source state inputs determine the states from which the transitions take place, while target state outputs determine the destination states of the fired transitions. The mode outputs are associated with a mode switch component in order to select the correct mode for execution.

3.4.3 Combining modes and state graphs

Once mode switch components and Gaspard state graph components are introduced, a `Macro Component` can be used to compose them together. An abstract representation of the macro component in Figure 4 illustrates one possible composition; and represents a complete Gaspard2 control structure. In the macro, the Gaspard state graph component produces mode values and sends them to the mode switch component. The latter switches the modes accordingly. Some data dependencies between these components are not always necessary, for example, data dependency between I_d and i_d, and these dependencies are drawn with dashed lines in Figure 4. The illustrated figure is used as a basic composition, however, other compositions are also possible, for instance, one Gaspard state graph component can control several mode switch components (Quadri et al. 2009*c*). In order to simplify the illustration, events `e1`, `e2` and `e3` are only shown as a single event I_e.

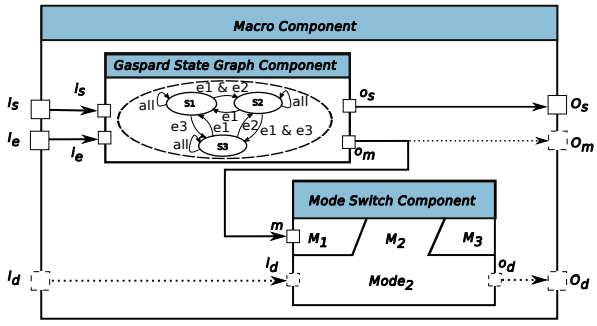


Figure 4: An example of a macro structure

3.5 MARTE concepts for constructing mode automata

We now present the utilization of some MARTE concepts which aid in the modeling of mode automata. The basic concepts of Gaspard2 control have already been presented earlier in section 3.4, but its complete semantics have not been detailed. For this purpose, we propose to integrate mode automata semantics with the control aspects. This choice is made to remove design ambiguity, enable desired properties and to enhance design correctness and verifiability. In addition to previously mentioned control concepts, we make use of three additional MARTE concepts, as present earlier in section 3.2; namely: `interRepetition` dependency, `tiler` and `defaultLink` connectors, which are essential for building mode automata.

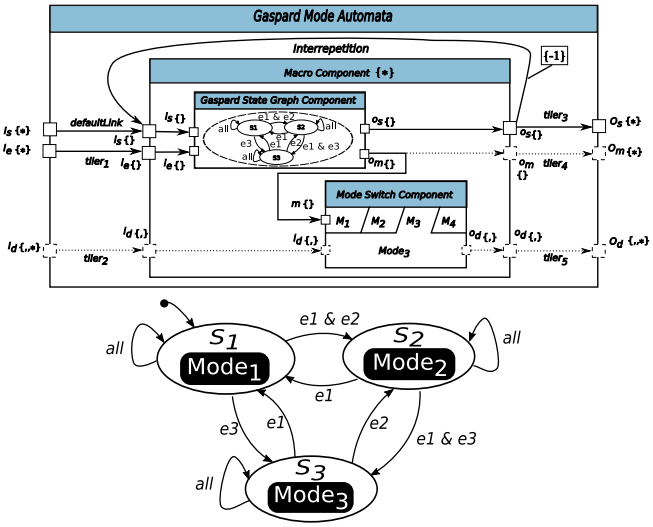


Figure 5: Abstract representation of a generic Gaspard2 mode automata (illustration on the top). The illustration on the bottom of the figure is a simplified explanation of the one on the top

Hence, it is possible to establish mode automata from Gaspard2 control model, which requires two subsequent steps. First, the internal structure of a generic `Gaspard Mode Automata` is presented by the `Macro component` illustrated in Figure 4. The Gaspard state graph component in the macro acts as a state-based controller and the mode switch component achieves the mode switch function. Secondly, `interRepetition` dependency specifications should be specified for the macro component and it should be placed in a repetitive component context.

The reasons are as follows: a macro component represents only one single transition function (one map) from a source state to a target state, where as an automata has continuous transitions which form an execution trace. In order to execute continuous transitions as present in a typical automata, the macro should be repeated to have multiple transitions. This functionality is determined by the `interRepetition` dependency.

A vector associated to an `interRepetition` dependency expresses the dependencies between the

repetitions inside the repetition context, i.e., the `Gaspard Mode Automata` component. Thus an `interRepetition` dependency serializes the repetitions and data can be conveyed between these repetitions. An `interRepetition` dependency sends the target state of one repetition as the source state to the next repetition. This permits the construction of mode automata which can be then executed. Figure 5 illustrates an example of this construction.

If a dependent repetition is not defined in the repetition space, a default value is selected. The `defaultLink` provides default value for repetitions whose dependency for the input is absent. Additionally, this concept helps to give the initial state value for the first repetition of the macro component. While in a graphical modeling approach, the initial state of a state machine/Statechart can be determined by an initial pseudostate, a Gaspard state graph does not contains an initial state, as explained earlier.

Thus this mechanism bridges the gap between a graphical representation and the actual semantics. It thus creates an equivalency between a state graph without no initial state and an automaton with an initial defined state. Finally, the `tiler` connectors help in interconnecting a repetitive component to the multiple repetitions of its interior repeated task.

An infinite dimension is present on the input and output state, events ports of the Gaspard mode automata component to account for continuous control/data flows. Similarly the non obligatory mode output ports, input and output data ports also have an infinite dimension in addition to other possible dimensions. Since the macro component represents one single transition, its respective ports have shape values equal to $\{\}$, accounting for one value in the dataflow at an instant of time t.

Similarly, the internal sub components of the macro component also share the same shape values. Finally, the shape value of $\{*\}$ on the macro component represents its multiple, possible infinite, dimensions. The macro component is repeated in a sequential temporal dimension by means of the `interRepetition` dependency. Thus as compared to traditional Statecharts, no final state is necessary. If the repetition of the macro component is not set to an infinite value, the macro will stop repeating when it reaches the end of its repetition space, causing termination of the mode automata.

It should be noted that parallel and hierarchical mode automata can also be built using our approach. In a parallel automata, several automatas can be executed in parallel, for example, one automata can be related to the application aspects while another can be linked to the architecture. Thus both application and architecture switches can be carried out simultaneously, provided that no conflicts arise by the simultaneous switching. Additionally, in case of control at a SoC design level such as application modeling, the switching between the states or modes can be instantaneous in nature, and is regarded as a change in the functionality. However when this semantics is applied to an execution platform, a stabilization phase may be required for switching between two states.

While this intermediate phase can be modeled using high level modeling semantics, this step has not been undertaken in our approach, in order to present a generic semantics applicable to all SoC design levels. Finally, we refer the reader to (Gamatié et al. 2008*a*) for the detailed formal semantics related to Gaspard mode automata.

4 Adaptivity at application level

The previous section described an abstract control model for integrating dynamic aspects in a given system. Similarly, these control mechanisms can be integrated in different levels in a SoC co-design framework, with the advantage of introducing dynamic aspects in the targeted SoCs. A detailed analysis related to control integration at different SoC design levels in the particular case of the Gaspard2 framework has been presented in (Quadri et al. 2009*c*). In the context of this article, we present the integration at the application and deployment modeling levels in Gaspard2.

4.1 Control example at application level

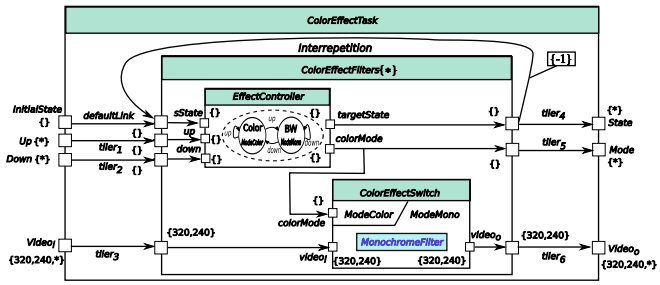


Figure 6: An example of color style filter in a smart phone modeled with the Gaspard2 mode automata

The control model enables the specification of system adaptivity at the application level (Yu 2008). Integration of control model and the construction of a mode automata at application level is very similar to the generic Gaspard mode automata shown in Figure 5. Figure 6 represents a mode automata at the application level by illustrating an example of color effect processing module `ColorEffectTask` used in typical smart phones. This module is used to manage the color effects of a video clip and provides two possible options: color or monochrome/black&white modes, which are implemented by `ColorFilter` and `MonochromeFilter` respectively. These two filters are elementary tasks at the application modeling level, which should be deployed to their respective IPs. The changes between these two filters are achieved by `ColorEffectSwitch` upon receiving mode values through its mode port `colorMode`. The mode values are determined by `EffectController`, whose behavior is demonstrated by its associated state graph.

The `ColorEffectFilters` can be treated as a macro component; and is composed of `EffectController` and `ColorEffectSwitch` components. `ColorEffectFilters` executes the processing of one frame of the video

clip, which should be repeated. In the example, `ColorEffectTask` provides the repetition context for `ColorEffectFilters`. An `interRepetition` dependency is also defined, which connects the different repetitions of the `ColorEffectFilters` component. It has an associated vector with a value of {-1}. Simply put, the source state of one `ColorEffectFilters` repetition relies on the target state of the previous `ColorEffectFilters` repetition. The data computations inside a mode are set in the mode switch component `ColorEffectSwitch`.

Each mode in the switch can produce different end results with regards to environmental or platform requirements. Each mode can have a different demand of memory, CPU load, etc. Thus environmental changes/platform requirements are captured as events; and taken as inputs for the control.

For the application level, the Gaspard2 control model has been implemented with UML state machines and collaborations in (Yu 2008). A model transformation chain from high level MARTE models to synchronous languages can bridge the gap between these models and targeted synchronous language code. By considering the code generated from an application model, validation techniques such as model checking can also be applied. The same code can also be used for controller synthesis to enforce relevant properties with respect to functional and non-functional requirements. All these aspects have been addressed in a case study for the design of a Gaspard2 data-parallel multimedia application (Yu 2008).

5 Adaptivity at IP deployment level

As explained before in the paper, we present an application driven approach for the design and development of dynamically reconfigurable SoCs. For this, we have focused on two main aspects related to a reconfigurable SoC.

For dynamic reconfiguration in modern SoCs, an embedded controller is essential for managing a dynamically reconfigurable region. This component is usually associated with some control semantics such as state machines, Petri nets etc. The controller normally has two functionalities: one responsible for communicating with the FPGA *Internal Configuration Access Port* hardware reconfigurable core or ICAP (Blodget et al. 2003) that handles the actual FPGA switching; and a state machine part for switching between the available configurations. The first functionality is written manually due to some low level technological details which cannot be expressed via a high level modeling approach.

The control at the deployment level is utilized to generate the second functionality automatically via model transformations. Finally the two parts can be used to implement partial dynamic reconfiguration in an FPGA that can be divided into several static/reconfigurable regions. A reconfigurable region can have several implementations, with each having the same interface, and can be viewed as a mode switch component with different modes. In our design flow, this dynamic region is generated from the high abstraction levels, i.e., a complex Gaspard2 application specified using the MARTE profile. Using the control aspects in the subsequently explained Gaspard2 deployment level, it is possible to create different configurations of the modeled application. Afterwards, using model transformations, the application can be transformed into a hardware functionality, i.e., a dynamically reconfigurable hardware accelerator, with the modeled application configurations serving as different implementations related to the hardware accelerator.

We now present integration of the control model at the deployment level. We first explain the deployment level in Gaspard and our extensions followed by the control model.

5.1 Deployment level in Gaspard2 framework

Gaspard2 defines a notion of a *Deployment* specification level (Atitallah et al. 2007) in order to generate compilable code from a SoC model. This level is related to the specification of *elementary* components: basic building blocks of all other components, having atomic functions. Although the notion of deployment is present in UML, the SoC design has special needs, not completely fulfilled by this notion. In order to generate an entire system from high level specifications, all implementation details of every elementary component have to be determined. Low level behavioral or structural details are much better described by using usual programming languages instead of graphical UML models.

Hence, Gaspard2 extends the MARTE profile to allow deploying of elementary components. The deployment level associates every elementary component to an implementation code hence facilitating IP reuse. Each elementary component ideally can have several implementations. The reason is that in SoC design, a functionality can be implemented in different ways. For example, an application functionality can either be optimized for a processor, thus written in assembler or C/C++, or implemented as a hardware accelerator using traditional HDLs like VHDL/Verilog or with SystemC. Hence the deployment level differentiates between the hardware and software functionalities; and permits moving from platform-independent high level models to platform dependent models for eventual implementation. Deployment provides IP information to model transformations to form a compilation chain, in order to transform the high abstraction level models for different domains: formal verification, simulation, high performance computing or synthesis. Hence deployment can be seen a potential extension of the MARTE profile enabling a complete flow from model conception to automatic code generation. We now present a brief overview of the deployment concepts.

A `VirtualIP` expresses the functionality of an elementary component, independently from the compilation target. For an elementary component *K*, it associates *K* with all its possible IPs. The desired IP is then selected by the SoC designer by linking it to *K* via an `implements` dependency. Finally, the concept of `CodeFile`, is used to specify, for a given IP, the file corresponding to the source code and its required compilation options. This last concept

is not illustrated in the paper due to space limitations. The `CodeFile` thus identifies the physical path of the source code. As compared to the deployment specified in (Atitallah et al. 2007), the current deployment level has been modified to respect the semantics of traditional UML deployment.

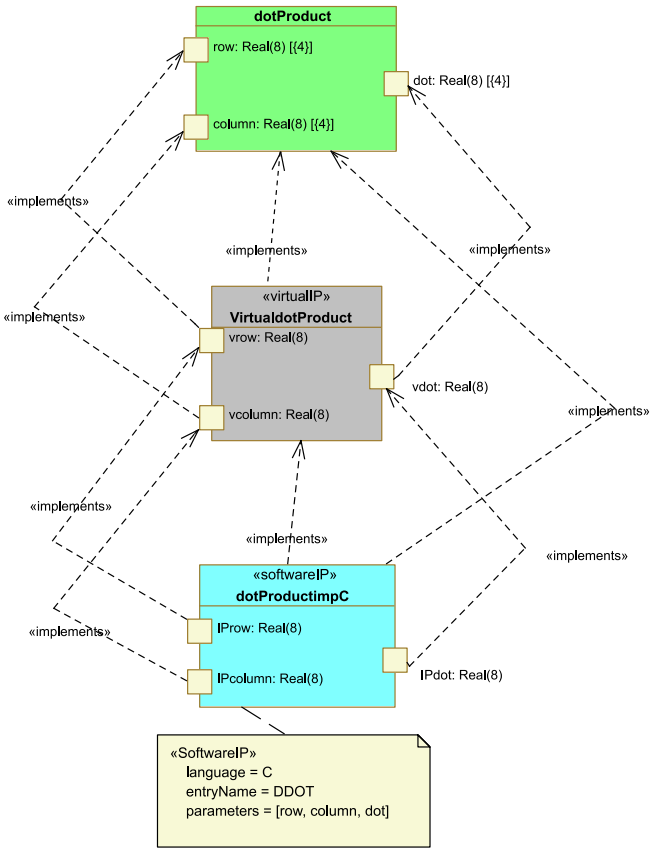


Figure 7: Deployment of an elementary `dotProduct` component in Gaspard2

5.2 *Configurations at the deployment level*

As stated before, an elementary component can be associated with only one IP among the different available choices. Thus the result of the application/architecture or the mapping of the two forming the overall system is a static one. This collective composition is termed as a *Configuration*. The current model transformations for RTL level only allow to generate one hardware accelerator from the modeled application, hence one configuration, for final FPGA execution.

Integrating control in deployment allows to create several configurations related to the modeled application for the final realization in an FPGA. Each configuration is viewed as a collection of different IPs, with each IP associated with its respective elementary component. The end result being that one application model is transformed by means of model transformations and intermediate metamodels into a dynamically reconfigurable hardware accelerator, having different implementations equivalent to the modeled application configurations.

A `Configuration` has the following attributes. The name attribute helps to clarify the configuration name given by a SoC designer. The ConfigurationID attribute permits to assign unique values to each `Configuration`, which in turn are used by the control aspects presented earlier. Theses values are used by a Gaspard state graph to produce the mode values associated with its corresponding Gaspard state graph component. These mode values are then sent to a mode switch component which matches the values with the names of its related collaborations, which are illustrated later on. If there is a match, the mode switch component switches to the required configuration. The InitialConfiguration attribute sets a Boolean value to a configuration to indicate if it is the initial configuration to be loaded onto the target FPGA. This attribute also helps to determine the initial state of the Gaspard state graph.

An elementary component can also be associated with the same IP in different configurations. This point is very relevant to the semantics of partial bitstreams, e.g., FPGA configuration files for partial dynamic reconfiguration, supporting *glitchless dynamic reconfiguration*: if a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation. If the same IP for an elementary component is present in several configurations, that IP is not changed during reconfiguration. It is thus possible to link several IPs with a corresponding elementary component; and each link relates to a unique configuration. We apply a condition that for any n number of configurations with each having m elementary components, each elementary component of a configuration must have *at least* one IP. This allows successful creation of a complete configuration for eventual final FPGA synthesis.

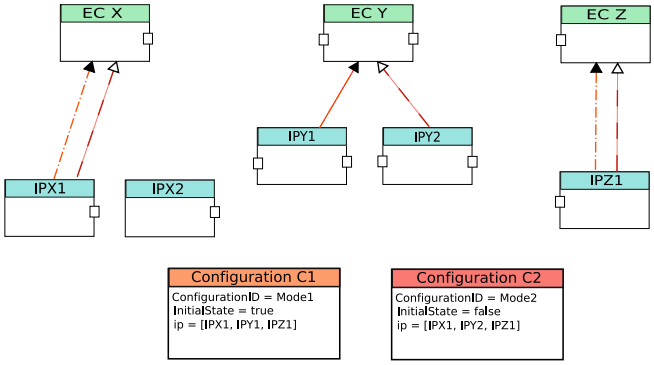


Figure 8: Abstract overview of configurations in deployment

Figure 8 represents an abstract overview of the configuration mechanism introduced at the deployment level. We consider a hypothetical Gaspard2 application having three elementary components *EC X*, *EC Y* and *EC Z*, having available implementations *IPX1*, *IPX2*, *IPY1*, *IPY2* and *IPZ1* respectively. For the sake of clarity, this abstract representation omits several modeling concepts such as `VirtualIP` and `Implements`. However, this representation is very close to UML modeling as presented earlier in the paper. A change in associated implementation of any of

these elementary components may produces a different end result related to the overall functionality, and different QoS criteria such as effectively consumed FPGA resources.

Here two configurations *Configuration C1* and *Configuration C2* are illustrated in the figure. *Configuration C1* is selected as the initial configuration and has associated IPs: *IPX1*, *IPY1* and *IPZ1*. Similarly *Configuration C2* also has its associated IPs. This figure illustrates all the possibilities: an IP can be globally or partially shared between different configurations: such as *IPX1*; or may not be included at all in a configuration, e.g., case of *IPX2*.

Once the different implementations are created by means of model transformations, each implementation is treated as a source for a partial bitstream. A bitstream contains packets of FPGA configuration control information as well as the configuration data. Each partial bitstream signifies a unique implementation, related to the reconfigurable hardware accelerator which is connected to an embedded controller. While this extension allows to create different configurations, the state machine part of the controller is created manually. For automatic generation of this functionality, the deployment extensions are not sufficient. We then use the existing control concepts presented in section 3.4 to solve these issues.

5.3 Integrating control at the deployment level

We now explain control integration at the deployment level in the context of the Gaspard2 SoC co-design framework. Control at this level provides advantages over other levels due to its independent nature. Details related to these advantages can be found in (Quadri et al. 2009*c*).

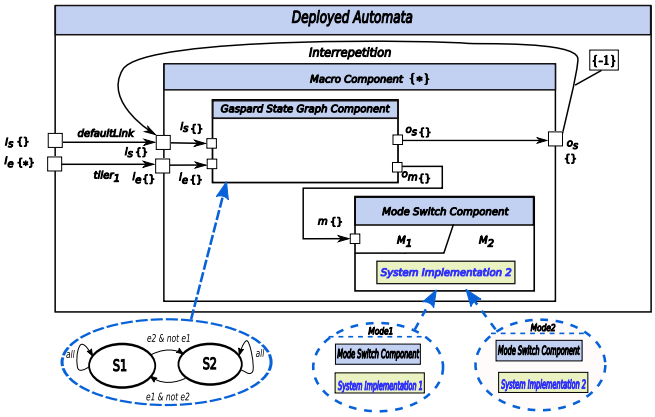


Figure 9: Integrating control at deployment level

Figure 9 shows the integration of control at the deployment level in Gaspard2. As compared to control models at other levels, e.g., such as application level, which only incorporate structural design aspects, this control model deals with behavioral aspects. The deployment level automata, termed as `Deployed Mode Automata` deals with atomic elementary components and their related implementations which are present at the lowest hierarchical level in the modeling; in order to address global system level implementations. As compared to other control models, a mode in a mode switch component represents a global system implementation of the modeled application, and is a collection of different implementations associated with their respective elementary components. Here, dataflow associated to the generic Gaspard mode automata is not explicitly expressed and input/output data ports are suppressed at all hierarchical levels in this control model.

Also we need to address the issue related to the incoming events arriving in a deployed mode automata. In a control model at application or architecture, the events arrive either from the external environment, or the events are produced at any time instant in the application or architecture itself due to the actions of some elementary components. However in the deployment level, the incoming events are not related to the high level modeling but are basically used to represent low level user inputs depending upon the chosen execution platform. For example at the RTL level, these user events can arrive in the form of user or environment input from a camera attached to an FPGA, or inputs received via an universal asynchronous receiver/transmitter or UART terminal. A designer modeling the system at a high level is not concerned with these low level implementation details. However, in order to make this control model as flexible as possible, and to respect the semantics of the abstract control model, event ports have been added to control-deployment proposal. During the model transformations and eventual code generation, these event ports are replaced and translated into actual event values which are used during FPGA implementation phase.

Similarly, for mode automaton at an application or architecture level, its initial state is usually determined by a component that has input event ports and an output state port. Initially some events are generated and taken as input by that component in order to produce the initiate state. After that, the component remains inactive due to the absence of the events arriving on its input ports. This initial state is then sent to the mode automata and serves to determine the initial state of the Gaspard state graph. However, for a deployed mode automata, structural aspects are absent and only information related to elementary components is present. Thus the initial state related to the deployed Gaspard state graph cannot be determined explicitly. This limitation has been removed by introducing new concepts at the deployment level, which help to determine the initial state of the deployed mode automata. However, the proposal retains the usage of an initial state port and the `defaultLink` concept, as they help to conform to the abstract control model; and are used in subsequent model transformations for eventual code generation.

Finally, the current control at deployment is only related to creating a state machine for a reconfigurable controller. In cases of FPGAs supporting several embedded hardcore/softcore processors; it is possible to select any one to act as a controller. However, this requires additional allocation types semantics to be linked to the deployment. Currently the code generated from our design flow is explicitly linked to a generic controller, and it is up to the user to determine the nature and position of the controller.

6 Case study

In order to validate our design flow, we now present a case study of an anti-collision radar detection system. Vehicle based anti-collision radar detection systems are becoming increasingly popular in automotive industry as well as in research. Furthermore, these devices provide additional safety to provide collisions and fatal accidents; and could become mandatory aboard vehicles in the next years. The principle of the system is to avoid collision between the equipped vehicle and the one in front, or other kind of obstacles such as pedestrians, animals. The algorithms which form the basis of these complex systems require large amounts of regular repetitive computations. This computational necessity requires the execution of these algorithms in parallel hardware architectures, such as hardware accelerators. We first provide a general overview of these systems, followed by the modeling of their key components and eventual code generation. Finally the paper provides implementation details for integrating aspects of dynamic reconfiguration in these systems.

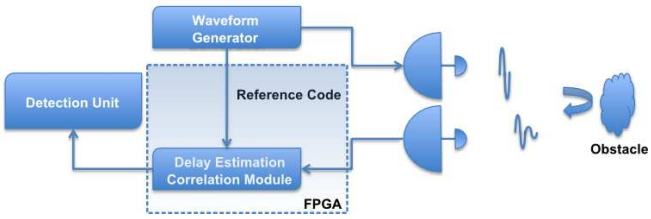


Figure 10: Block diagram of the anti-collision radar detection system

The anti-collision radar detection system is illustrated in Figure 10. The radar consists of two antennas; and emits a signal modulated with a *Pseudo Random Binary Sequence* (PRBS), resulting in formation of a reference code (Quadri et al. 2009*a*). The PRBS has interesting correlation as well intercorrelation characteristics (Douadi et al. 2008). When the transmitted wave encounters an obstacle, it is reflected and creates an echo which is captured by means of the second antenna. The echo is converted into a signal containing information related to the distance of the detected obstacle. Unfortunately, this information cannot be directly interpreted due to the presence of time delays and noise in the incoming signal. The PRBS present in the incoming signal is recognized by means of a *Delay estimation correlation module* or DECM present in the embedded system; and determines the time of flight. Thus, distance to the object and its speed can be calculated easily.

Also, it is not mandatory to exploit all the precision of the returned signal, since the information contained in the least significant bits is embedded with noise. In (Douadi et al. 2008), the authors recommend to use only 4 bits of the incoming signal, because the information contained in the 5th and the following bits are not significant.

For the radar system, the DECM can be implemented on an FPGA, as these reconfigurable SoCs allow to execute the detection algorithm that retains the necessary information present in the incoming signal. This information corresponds to the PRBS utilized in the emission of the signal. The role of the detection algorithm is to highlight the similarities between the reference code and the received signal: when the received signal corresponds with the reference code, the presence of an obstacle is detected. The inverse case means that the received signal contains little or no information related to the reference code and therefore, objects are not effectively detected.

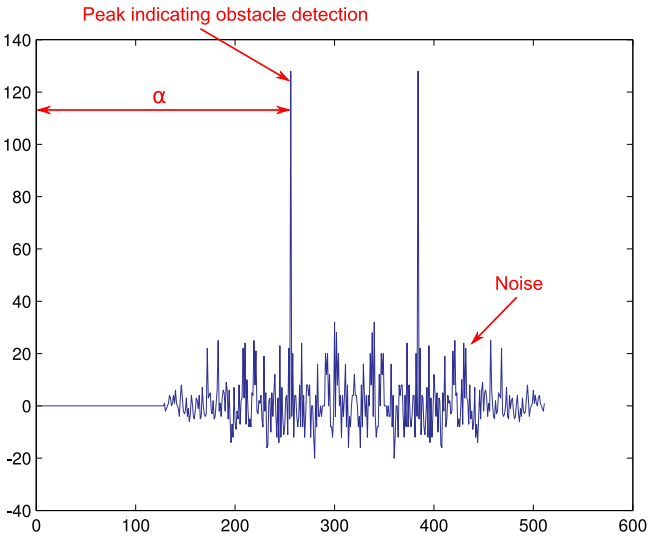


Figure 11: MATLAB result of the correlation

Figure 11 shows the result of a simulated correlation measurement in MATLAB. The outcome of a correlation between the reference code of a 127 length PRBS and the received simulated response (integrated with time delays and noise) yields a peak as indicated in the figure. A peak indicates the successfull detection of an obstacle, and its position corresponds to the delay that we have introduced in the simulation. As the radar emits and receives a signal continuously in a temporal dimension, the correlation step is also repeated continuously, resulting in a peak at different intervals of time, indicating object detection at different time intervals. In the Figure, we illustrate the results of two correlations. To perform the necessary obstacle detection with the radar, the correlation peaks need to be localized, between the emitted code and its returned echo. The object which is detected by the correlation has a distance d from the radar, which is given by:

$$d = c\alpha/2 \tag{1}$$

Where c is the speed of the propagated signal (equal to 3.10^8, corresponding to the speed of light); and α is the respective time delay.

In this section, we have presented the structure of the anti-collision radar detection system. The DECM module is the key element of this radar detection system, and the correlation computation is very time consuming especially for longer PRBSs. Our case study is mainly concerned with this functionality; details related to the modeling of

the DECM module have been presented in (Quadri et al. 2009*a*,*d*), hence in the context of this paper, we only provide the top hierarchical level of our modeled application.

6.1 Delay estimation correlation module

Correlation algorithms are among the type of digital processing largely employed in DSP (digital signal processing) based systems. They offer a large applicability range such as linear phase and stability. A correlation algorithm normally takes some input data values and computes an output which is then multiplied by a set of coefficients. Afterwards the result of this multiplication is added together to produce the final output. While a software implementation can be utilized for implementing this functionality, the correlation functionality will be sequentially executed. Where as a hardware implementation allows the correlation functions to be executed in a parallel manner and thus increases the processing speed. However this implementation is not flexible for minute changes, hence a reconfigurable DECM module is an ideal solution as it offers the flexibility of a software implementation while retaining the capability to construct customized high performance computing circuits.

Figure 12 represents the top level of our modeled DECM module. The component instance `trm` of the component `TimeRepeatedMultiplicationAddition` determines the global multiplications while instance `trat` of component `TimeRepeatedAdditionTree` determines the overall sum. The `TimeRepeatedMultiplicationAddition component` itself carries out a partial sum between received elements of the reference code and the received signal at each clock cycle, which are then sent to the `TimeRepeatedAdditionTree` component to execute the overall addition operation. The instance `trdg` of component `TimeRepeatedDataGen` produces the data values for the generated incoming signal while the instance `trcg` of component `TimeRepeatedCoeffGen` produces the reference code.

We have identified four key elementary components `CoeffGen`, `DataGen`, `MultiplicationAddition` and `Addition` in our modeled application as shown in Figure 12. They are present in different levels of hierarchy in the components `TimeRepeatedCoeffGen`, `TimeRepeatedDataGen`, `TimeRepeatedMultiplicationAddition` and `TimeRepeatedAdditionTree` respectively at the top level of the application in Figure 12. Any change in the implementations of any of the elementary component directly affects the final result as well as other QoS criteria: such as as reconfiguration time; consumed FPGA resources and the computation power. Deployment of these elementary components such as that of `MultiplicationAddition` can effect the overall QoS results. While it is theoretically possible to have a large number of configurations, only two have been considered for our case study. In this paper, we propose to associate two different implementations related to the `MultiplicationAddition` component, one written in a DSP like fashion, while other written using an If-then-else construct. While changing of an IP related to an elementary component might seem insignificant, it causes a global influence resulting in different QoS end results related to the DECM module.

Afterwards, the deployment phase is carried out as illustratred as abstractly represented in Figure 13 and all the elementary components are deployed. Then the modeling of the mode automata related to the DECM is initiated; with the mode automata serving to switch between the different DECM configurations.

Figures 14 and 15 illustrate the various concepts related to the construction of the mode automata. This modeling approach corresponds to the abstract control concepts introduced earlier in the paper, thus redundant explanatory information is unnecessary. Here the `DECM State Graph` contains two states `state_DECM_DSP` and `state_DECM_Ifelse` corresponding to the respective configurations modeled previously. This state graph is related to the `DECM State Graph Component` that serves as a control component. Its counterpart, the controlled mode switch component or `DECM MSC`, contains several collaborations, each signifying the internal behavior of this mode switch components on the basis of their interior parts and the incoming mode value on the port of the mode switch component. The combination of the control and controlled component forms the basis of a macro component that represents a single transition in the mode automata. For continuous transitions, the macro is placed in a repetitive context task: the `DECM Mode Automata` component along with its respective `tilers`, `interRepetition` and `defaultLink` dependencies.

As mentioned previously, one of goals of our design flow is the creation of a dynamically reconfigurable hardware accelerator with several configurations, that can be swapped dynamically in a run time reconfigurable SoC. The UML model is transformed by the various model transformations present in our design methodology, they generate the various implementations related to the hardware functionality. The control model is equivalently converted into a state machine for eventual utilization by the reconfiguration controller of the SoC in question. Figure 16 represents the Gaspard2 framework in the Eclipse environment as well as the different models present in our design flow. The *UML model* corresponds to the modeled control integrated deployed functionality and is directly generated from the UML diagram with integrated MARTE profile. The model-to-model transformations mentioned earlier permit to create several intermediate models, such as the RTL model corresponding to its own metamodel. Finally, the last step of our design flow consists of the generation of the source code related to the hardware accelerator and the configuration controller, by means of a model-to-text transformation. We now present some of the simulation results related specifically to the generated hardware functionality.

6.2 Simulation of the modeled functionality

The verification of the modeled application and its eventual equivalent hardware execution is first carried out by means

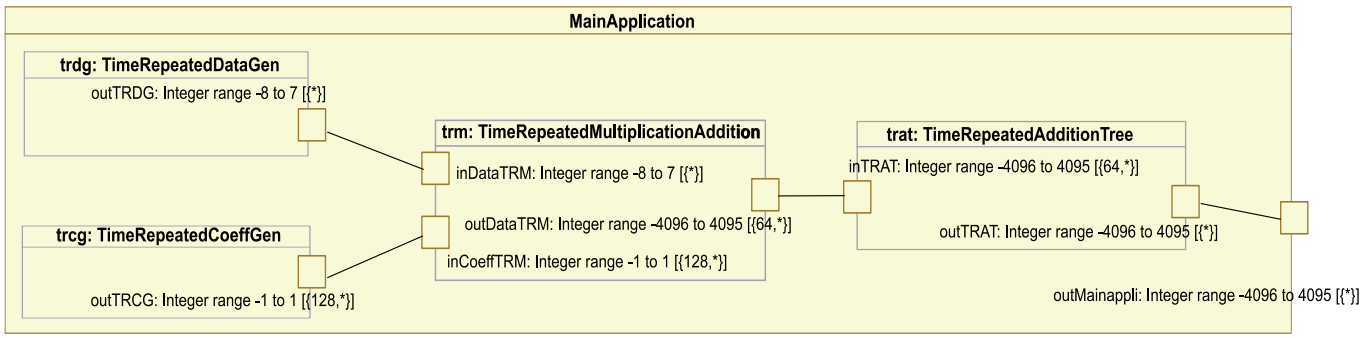


Figure 12: The top level view of the DECM

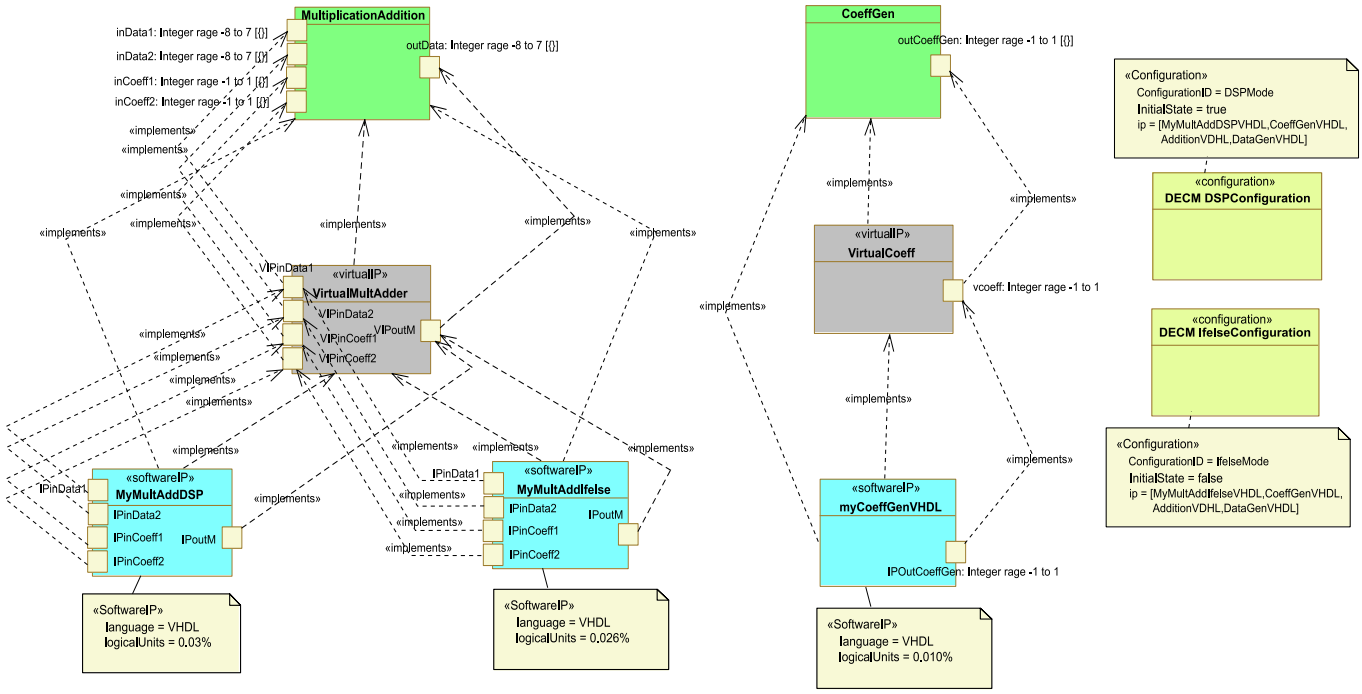


Figure 13: Deployment of the elementary components of the DECM

of simulation using the industry standard ModelSim[6] simulation tool. Once the code for the various configurations has been generated from the model transformations, we move on to the simulation phase. The simulation helps to verify the correctness of the generated functionality, before moving on to the eventual FPGA synthesis. Figures 17 and 18 show the simulation results of the DSP configuration. Here in the simulation, we observe several intermediate integer values ranging between -100 to 200, between time intervals of 20 ns. These values are equivalent to the noise illustrated in the MATLAB simulation results. The simulation is set to run for about 8000 ns which is sufficient to observe the execution time of the application functionality. In the simulation, besides the intermediate noise values, we observe two significant integer values of more than 900 at two distinct time instants.

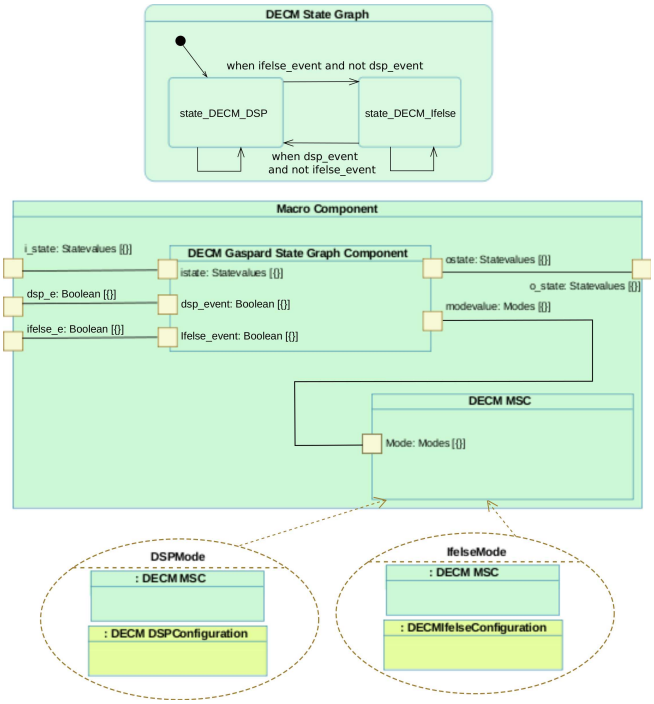


Figure 14: Mode automata concepts for the DECM: part one

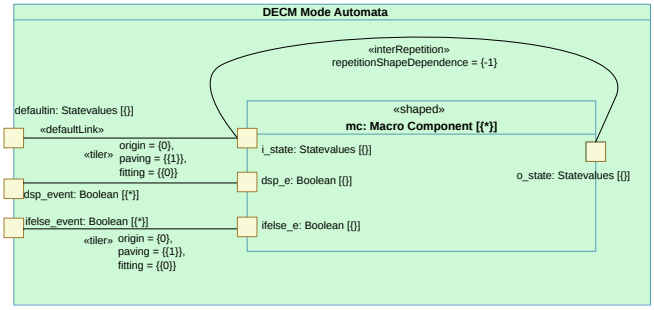


Figure 15: Mode automata concepts for the DECM: part two

These two values are equivalent to two peaks in the simulation, which correspond to the MATLAB simulation result illustrated in Figure 11. As the simulation results are a perfect match to the earlier MATLAB simulation results, the generated configuration is considered valid. The simulation results verify the functionality related to the different implementations of the high level modeled application functionality.

Currently in our design flow, we primarily make use of simulation to verify the correctness of the generated code. However, it may be possible that the generated code does not produce expected results or the generated syntax was incorrect. As a solution, model based techniques such as traceability can be of extreme significance. Traceability helps to determine errors in the high level models and the corresponding model transformations. Currently this aspect is being studied in our research team for eventual integration in the Gaspard2 framework, and could be a future extension of our design flow (Aranega et al. 2009).

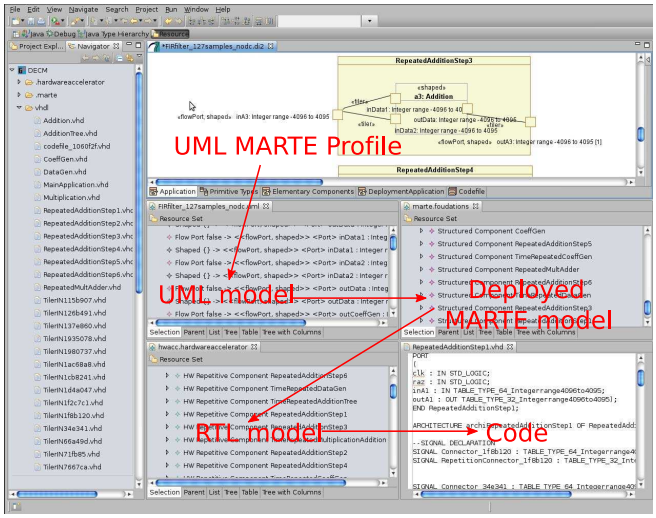


Figure 16: The transformation flow related to our design flow

Figure 17: First peak of the DSP configuration

Figure 18: Second peak of the DSP configuration

6.3 Implementing a partial dynamically reconfigurable DECM

In the previous sections, we have presented the initial details related to the application selected for this paper,

along with its modeling at the MARTE profile level. Afterwards via the design flow presented during the course of this paper, the integrated model transformations generate the source code from the high abstraction level input models. Once the source code has been generated, we move onto implementing a partial dynamically reconfigurable SoC (Quadri et al. 2009*d*). This section deals with the implementation details and provides the validation of our design methodology.

We first investigated the architecture related to implementing partial dynamic reconfiguration in Xilinx FPGAs. In Figure 19 we present the global structure of our reconfigurable architecture that was implemented on the Xilinx Virtex-II Pro XC2VP30 FPGA on a XUP Board[7]. This particular type of structure is popular in the domain related to dynamic reconfigurable FPGAs, and various variants have been built from this classical structure, such as presented in (Claus et al. 2007, Tumeo et al. 2007). The choice of selecting the classical structure was 1) to compare our system with other existing partial dynamic reconfiguration based systems in literature, and 2) to provide the basic template for a model driven dynamically reconfigurable system that can be optimized by the domain experts, in order to generate their customized versions.

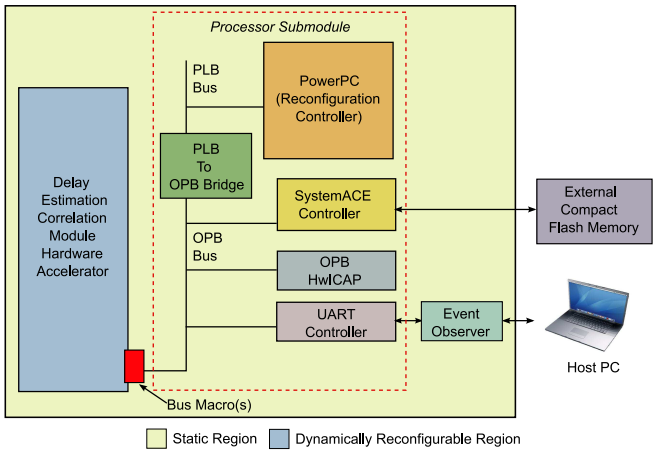


Figure 19: Block Diagram of the architecture of our reconfigurable system

In our selected system structure, we make use of the embedded hardcore PowerPCs present in the Xilinx Virtex-II Pro series FPGAs. One of the PowerPCs is selected as the reconfigurable controller and the state machine code generated from the high level control model in our design flow is executed on this processor. The partial dynamic reconfiguration system can be mainly divided into two main parts. The static region and the dynamically reconfigurable one. The static region mainly consists of a processor sub module that contains the reconfiguration controller and other necessary peripherals for dynamic reconfiguration.

The processor submodule is connected to a dynamically reconfigurable hardware accelerator via bus macros. These bus macros are communication modules that help in the communication between the static/dynamic regions. The hardware accelerator is equivalent to the hardware functionality generated from the high level modeled application in our design flow, and serves as the partially reconfigurable region in the overall system. The various implementations/partially reconfigurable modules related to the partially reconfigurable region are consistent with the modeled configurations at the deployment phase. The bus macros which are connected to the outputs of the hardware accelerator have a special enable/disable signal that permits the controller to disable the bus macros during a configuration switch to another state. Once a successful switch is carried out, the bus macros are enabled again. Thus during the switch, no output is generated from the partially reconfigurable region, causing the system to always remain in a safe state.

Finally the processor submodule system is connected to an event observer. The event observer receives the event values and relays them to the RS232 UART (universal asynchronous receiver/transmitter) controller of the processor submodule. Users can send input events, from the host PC, related to the configuration switches to the partial dynamic reconfiguration system, by means of a hyperTerminal. A program running on the hyperTerminal gives the user the choice of switching between the available configurations. Each configuration switch is related to a specific input character, that is mapped to a specific event in the executing controller program. When this value is received by the partial dynamic reconfiguration system, the associated state transition is carried out.

Once the code has been generated from our model driven design flow, we move on to the initial design partition phase of our partial dynamic reconfiguration system according to the Xilinx EAPR flow (Xilinx 2006). The processor submodule for the partial dynamic reconfiguration system is initially created by means of the Xilinx Platform studio. The source code for the controller is selected to be executed on the PowerPC 405_0, with a clock frequency of 100 MHz. The second PowerPC while present in the figure, is not connected to any clock signals and is therefore deactivated. A Processor Local Bus Block-RAM or PLB BRAM interface controller permits interfacing between the PLB and a Block-RAM of size 128 KB. This size is sufficient to store the data and instructions of the executable processor code, and on-chip-memory is not required. Using FPGA BRAMs to store the data/instructions allows the processor code and initialized variables to be written directly into the memory, when the FPGA is configured initially.

The processor subsystem is then inserted into a top level VHDL file that contains the component instantiations and port mappings related to the processor subsystem and the dynamically reconfigurable DECM module. This DECM module is connected to the static processor subsystem by means of bus macros which are also present in the top hierarchical level. Afterwards, synthesis is carried out to generate the appropriate files for eventual implementation of Partial dynamic reconfiguration using the PlanAhead design tool (Xilinx 2006).

We now present the partial synthesis results of some of the modeled application components in our case study carried out with the Xilinx ISE on the XUP board. Figure 20

shows the global view of the synthesis of the modeled DECM application.

Once the synthesis has been carried out, we move onto generation of the partial bitstreams related to the different modeled configurations as well as the static bitstream. Figure 21 only shows the partial bitstream related to the DSP configuration, while Figure 22 shows the full initial bootup bitstream that is a merge of the static bitstream and the DSP partial bitstream.

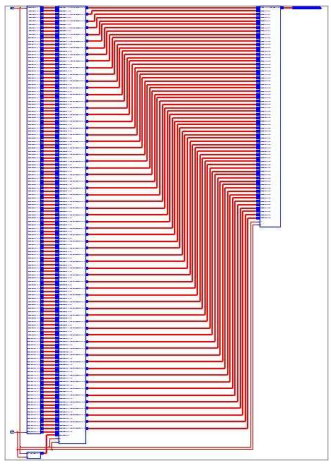


Figure 20: Synthesis result of the top level of the DECM

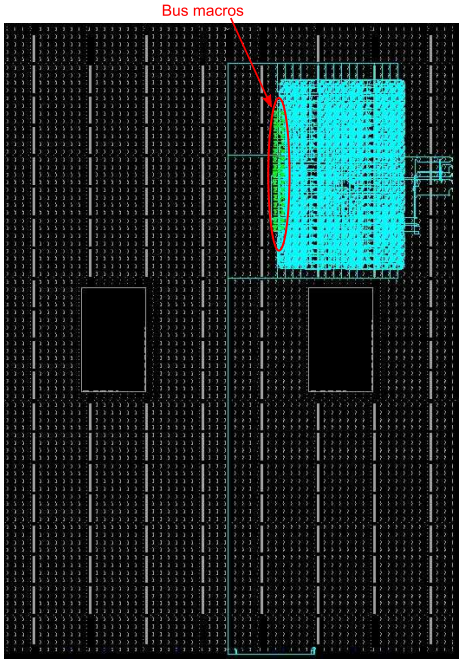


Figure 21: Partial bitstream related to the DSP configuration

Table 1 shows the results related to the two configurations. The first configuration consumes slightly more FPGA resources as compared to the second one. Additionally, the reconfiguration time for the first configuration is higher as compared to the second one. This is due to the fact that the ICAP core needs to modify several additional frames for the first configuration, as compared to the latter. While the reconfiguration time is extremely high for both configurations, this is due to the low bandwidth: 115200 bps; of the RS232 controller and the large size of the partial bitstreams. Using an external RAM memory can greatly increase the reconfiguration times, similarly various other optimizations can be carried out with respect to this implementation, such as introducing a DMA in the reconfigurable system, a customized ICAP controller or usage of a PLB ICAP core.

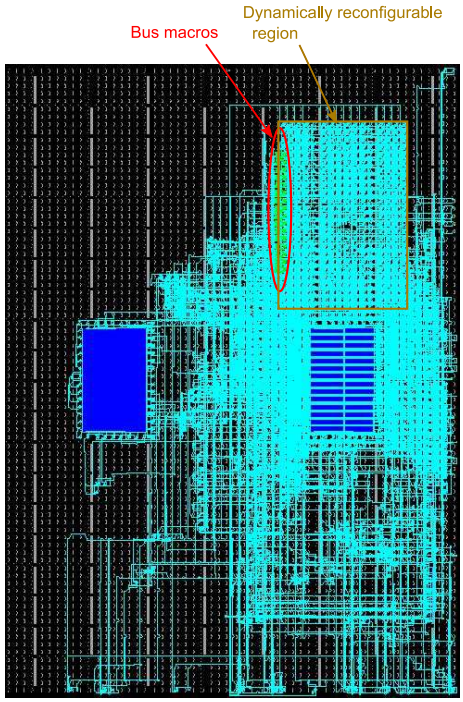


Figure 22: Full bitstream related to the partial dynamic reconfiguration system

	DSP Configuration	If-then-else Configuration
Slices	1272/13696 (9.287%)	1186/13696 (8.659%)
Slice FlipFlops	2084/27392 (7.608%)	1944/27392 (7.096%)
LUTs	1584/27392 (5.782%)	1836/27392 (6.702%)
Time (secs)	1.45	1.41

Table 1 Results related to the two configurations for the hardware accelerator. The percentage is in overview of the total FPGA resources.

7 Control models and FPGA synthesis

Many different approaches exist for expressing control semantics, such as Petri Nets (Nascimento et al. 2004); *if-then-else*, *switch* and *goto*-based semantics. However mode automata were selected as they clearly separate control/data flow. They also adapt a state based approach facilitating seamless integration in our framework; and can be expressed at the MARTE specification levels. The same

control semantics are then used throughout our framework to provide a single homogeneous approach.

With regard to partial dynamic reconfiguration, different implementations of a reconfigurable region must have the same external interface for integration with the static region at run-time. Mode automata control semantics can express the different implementations collectively via the concept of a mode switch, which can be expressed graphically at high abstraction levels using the concept of a mode switch component. Similarly a state graph component expresses the controller responsible for the context switch between the different implementations.

Both control models expressed in this paper can be used for FPGA synthesis. The first model introduces dynamic aspects in the application, which may modify the structure of an application, as different modes in a mode switch component can have different natures. For example, one mode could be elementary in nature while another can be hierarchically composed. The application can be associated and correspondingly deployed in different ways. The whole application could be allocated and deployed onto a single processor or an hardware accelerator; or split into parts. The first case while allowing reconfiguration, is not dynamically reconfigurable in nature.

In the second case, the parts can be correspondingly allocated and deployed: the state graph component onto a processor, and the mode switch components onto the reconfigurable regions. The disadvantage of this approach is that multiple allocations are required between the application and the targeted architecture. This control model can also be applied onto the architecture level of our framework. A processor can have a reconfiguration manager observing QoS criteria. If the processor heats up during execution, the manager can change the frequency of the processor. An internal controller could carry out aspects of dynamic reconfiguration. In addition, an external controller can be used to globally change the architecture.

The deployment level control model renders the application or architecture reusable. The designer can change partial functionality of the application by changing some IPs related to corresponding elementary components. This model is thus more interesting as one application functionality can be reused without changing its structure, and its overall implementation can be changed only depending upon QoS criteria and hardware resources limitations. Currently the deployment level is explicitly linked with a specific reconfigurable controller in the targeted FPGA. In case of multiple processors: one managing the configuration and the other executing some application functionality, this information must be elevated to the allocation level to correctly link the related entities. It is also possible to combine the two control models for simultaneous integration in our framework.

Additionally, some of the low level RTL details can be integrated into high level models by means of a *Design Space Exploration* strategy. For example, reconfiguration time related to a specific configuration can be determined after an initial synthesis and then can be added to models by means of UML attributes. This will allow designers to take QoS criteria into consideration when changing between the different configurations. Additionally, details related to the targeted architecture can also be modeled at the high abstraction levels. An initial contribution related to this aspect has been detailed in (Quadri et al. 2009*b*); and could be a future extension of our design flow.

8 Conclusion

This paper presented an high level design flow for targeting reconfigurable SoCs in the context of a model-driven co-design framework, Gaspard2, which is compliant with the MARTE standard. We have selected two key points of these reconfigurable systems to be modeled at high abstraction levels. We have mainly taken into account the dynamically reconfigurable region and the semantics related to the reconfigurable controller; managing for switching between the different implementations related to the region. The control semantics is based on mode automata and is integrated at different levels of SoC co-design. In the context of partial dynamic reconfiguration, they have been integrated at the deployment level in our Gaspard2 framework. Afterwards, a case study consisting of a component in an anti-collision radar detection system has been illustrated for validating the proposed design methodology. Our high-level modeling approach enables implementation and FPGA synthesis of various system configurations rapidly in a flexible manner. It therefore helps to explore different design choices about the system, which is usually a delicate task. Finally, we provide a comparison between the control semantics at different design levels, specifically for FPGA synthesis and dynamic reconfiguration.

References

Aranega, V., Mottu, J.-M., Etien, A. & Dekeyser, J.-L. (2009), Traceability Mechanism for Error Localization in Model Transformations, *in* '4th International Conference on Software and Data Technologies (ICSOFT 2009)', Sofia, Bulgaria. To Appear.

Atitallah, R. B., Piel, E., Niar, S., Marquet, P. & Dekeyser, J.-L. (2007), Multilevel MPSoC simulation using an MDE approach, *in* 'SoCC 2007'.

Bayar, S. & Yurdakul, A. (2008), Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP), *in* 'HiPEAC'08 Workshop on Reconfigurable Computing'.

Becker, J., Huebner, M. & Ullmann, M. (2003), Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations, *in* 'VLSI'03'.

Bergmann, N., Williams, J. & Waldeck, P. (2003), Egret: A flexible platform for real-time reconfigurable system-on-chip, *in* 'Proceedings of International Conference on Engineering Reconfigurable Systems and Algorithms', pp. 300–303.

Blodget, B., McMillan, S. & Lysaght, P. (2003), A lightweight approach for embedded reconfiguration of FPGAs, *in* 'Design, Automation & Test in Europe, DATE'03'.

Boulet, P. (2007), Array-OL revisited, multidimensional intensive signal processing specification, Research Report RR-6113, INRIA, http://hal.inria.fr/inria-00128840/en/.

Boulet, P. (2008), Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing, Technical Report 6467, INRIA, France. http://hal.inria.fr/inria-00261178/en.

Brito, A., Kuhnle, M., Hubner, M., Becker, J. & Melcher, E. (2007), Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC, *in* 'Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)', IEEE Computer Society, pp. 35–40.

Claus, C., Muller, F., Zeppenfeld, J. & Stechele, W. (2007), 'A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration', *IPDPS 2007* pp. 1–7.

Douadi, L., Deloof, P. & Elhillali, Y. (2008), Real time implementation of reconfigurable correlation radar for road anticollision system, *in* 'IEEE International Conference on Industrial Technology (ICIT 2008)', pp. 1–7.

Ebeling, C., Cronquist, D. & Franklin, P. (1996), Rapid - reconfigurable pipelined datapath, *in* 'Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers (FPL' 96)', Springer-Verlag, pp. 126–135.

F. Berthelot and F. Nouvel and D. Houzet (2008), 'A Flexible system level design methodology targeting run-time reconfigurable FPGAs', *EURASIP Journal of Embedded Systems* **8**(3), 1–18.

Faugere, M., Bourbeau, T., Simone, R. & Sebastien, G. (2007), MARTE: Also an UML Profile for Modeling AADL Applications, *in* 'ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems', IEEE Computer Society, pp. 359–364.

Gamatié, A., Le Beux, S., Piel, E., Ben Atitallah, R., Etien, A., Marquet, P. & Dekeyser, J.-L. (2010), 'A model driven design framework for massively parallel embedded systems', *ACM Transactions on Embedded Computing Systems (TECS)*. To appear.

Gamatié, A., Rutten, E. & Yu, H. (2008*a*), A Model for the Mixed-Design of Data-Intensive and Control-Oriented Embedded Systems, Research Report RR-6589, INRIA, http://hal.inria.fr/inria-00293909/fr.

Gamatié, A., Rutten, E., Yu, H., Boulet, P. & Dekeyser, J.-L. (2008*b*), 'Synchronous Modeling and Analysis of Data Intensive Applications', *EURASIP Journal on Embedded Systems*. Hindawi Publishing Corporation.

Harel, D. (1987), 'Statecharts: A Visual Formalism for Complex Systems', *Science of Computer Programming* **8**(3), 231–274.

INRIA Atlas Project (n.d.), 'ATL', http://modelware.inria.fr/rubrique12.html.

INRIA DaRT team (2009), 'GASPARD SoC Framework'. http://www.gaspard2.org/.

INRIA Triskell Project (n.d.), 'Kermeta', http://www.kermeta.org/.

J.P. Diguet and G. Gogniat and J.-L. Philippe et al (2006), 'EPICURE: A partitioning and co-design framework for reconfigurable computing', *Journal of Microprocessors and Microsystems* **30**(6), 367–387.

Koch, R., Pionteck, T., Albrecht, C. & Maehle, E. (2006), An adaptive system-on-chip for network applications, *in* 'IPDPS 2006'.

Koudri, A., Aulagnier, D., Vojtisek, D., Soulard, P., Moy, C., Champeau, J., Vidal, J. & Lann, J.-C. (2008), Using MARTE in the MOPCOM SoC/SoPC Co-Methodology, *in* 'MARTE Workshop at DATE'08'.

Labbani, O., Dekeyser, J.-L., Boulet, P. & Rutten, E. (2005), Introducing control in the Gaspard2 Data-Parallel MetaModel: Synchronous Approach, *in* 'Proceedings of the International Workshop MARTES: Modeling and Analysis of Real-Time and Embedded Systems'.

Latella, D., Majzik, I. & Massink, M. (1999), Automatic Verification of a Behavioral Subset of UML Statechart Diagrams Using the SPIN Model-Checker, *in* 'Formal Aspects Computing', Vol. 11, pp. 637–664.

Le Beux, S. (2007), Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles, PhD thesis, University of Lille 1, France.

Lysaght, P., Blodget, B. & Mason, J. (2006), Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs, *in* 'FPL'06'.

Maraninchi, F. & Rémond, Y. (2003), 'Mode-automata: a new domain-specific construct for the development of safe critical systems', *Sci. Comput. Program.* **46**(3), 219–254.

Moore, G. (1965), 'Cramming more components into integrated circuits', *Electronics* **38**(8), 114–117.

Morpheus (2010), 'Multi-purpose dynamically reconfigurable platform for intensive heterogeneous processing'. http://www.morpheus-ist.org/.

Nascimento, B., Sérgio, P., Maciel, M., Romero, P., Lima, M., Sant'ana, R., Filho, S. & Guilhermino, A. (2004), A partial reconfigurable architecture for controllers based on Petri nets, *in* 'SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design', pp. 16–21.

Nezami, K. G., Stephens, P. W. & Walker, S. D. (2008), Handel-C Implementation of Early-Access Partial-Reconfiguration for Software Defined Radio, *in* 'IEEE Wireless Communications and Networking Conference (WCNC'08)', pp. 1103–1108.

OMG (2005), 'MOF Query /Views/Transformations'. http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

OMG (2007*a*), 'M2M/Operational QVT Language'. http://tiny.cc/tFGGx.

OMG (2007*b*), 'Portal of the Model Driven Engineering Community'. http://www.planetmde.org.

OMG (2008), 'Modeling and Analysis of Real-time and Embedded systems (MARTE)', http://www.omgmarte.org/.

Panainte, E. M., Bertels, K. & Vassiliadis, S. (2007), 'The molen compiler for reconfigurable processors', *ACM Transactions in Embedded Computing Systems (TECS)*.

Paulsson, K., Hubner, M., Auer, G., Dreschmann, M., Chen, L. & Becker, J. (2007), 'Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGA', *FPL 2007* pp. 351–356.

Pillement, S. & Chillet, D. (2009), High-level model of dynamically reconfigurable architectures, pp. 1–7.

Quadri, I. R., Boulet, P., Meftali, S. & Dekeyser, J.-L. (2008), Using an mde approach for modeling of interconnection networks, *in* 'The International Symposium on Parallel Architectures, Algorithms and Networks Conference (ISPAN 08)'.

Quadri, I. R., Elhillali, Y., Meftali, S. & Dekeyser, J.-L. (2009*a*), Model based design flow for implementing an Anti-Collision Radar system, *in* '9th International IEEE Conference on ITS Telecommunications (ITS-T 2009)'.

Quadri, I.-R., Meftali, S. & Dekeyser, J.-L. (2009*b*), 'A Model Driven design flow for FPGAs supporting Partial Reconfiguration', *International Journal of Reconfigurable Computing*. Hindawi Publishing Corporation.

Quadri, I. R., Meftali, S. & Dekeyser, J.-L. (2009*c*), Integrating Mode Automata Control Models in SoC Co-Design for Dynamically Reconfigurable FPGAs, *in* 'International Conference on Design and Architectures for Signal and Image Processing (DASIP 09)'.

Quadri, I. R., Muller, A., Meftali, S. & Dekeyser, J.-L. (2009*d*), MARTE based design flow for Partially Reconfigurable Systems-on-Chips, *in* '17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)'.

S. Sendall and W. Kozaczynski (2003), 'Model Transformation: The Heart and Soul of Model-Driven Software Development', *IEEE Software* **20**(5), 42–45.

Salefski, B. & Caglar, L. (2001), Reconfigurable computing in wireless, *in* 'Proceedings of the 38th annual Design Automation Conference (DAC'01)', ACM, pp. 178–183.

Schäfer, T., Knapp, A. & Merz, S. (2001), Model Checking UML State Machines and Collaborations, *in* 'CAV Workshop on Software Model Checking', ENTCS 55(3).

Schuck, C., Kuhnle, M., Hubner, M. & Becker, J. (2008), A framework for dynamic 2D placement on FPGAs, *in* 'IPDPS 2008'.

Sedcole, P., Blodget, B., Anderson, J., Lysaght, P. & Becker, T. (2005), Modular Partial Reconfiguration in Virtex FPGAs, *in* 'FPL'05', pp. 211–216.

SPEEDS! (2009), 'SPEculative and Exploratory Design in Systems Engineering'. `http://www.speeds.eu.com/`.

Tumeo, A., Monchiero, M., Palermo, G., Ferrandi, F. & Sciuto, D. (2007), 'A Self-Reconfigurable Implementation of the JPEG Encoder', *ASAP 2007* pp. 24–29.

Vidal, J., Lamotte, F. D. & Gogniat, G. (2009), A co-design approach for embedded system modeling and code generation with UML and MARTE, *in* 'Design, Automation and Test in Europe (DATE'09)'.

Xilinx (2006), Early Access Partial Reconfigurable Flow. `http://www.xilinx.com/support/prealounge/protected/index.htm`.

Yu, H. (2008), A MARTE-Based Reactive Model for Data-Parallel Intensive Processing: Transformation toward the Synchronous Model, PhD thesis, USTL/LIFL, France.

Yu, H., Gamatié, A., Rutten, E. & Dekeyser, J.-L. (2008), 'Safe Design of High-Performance Embedded Systems in a MDE framework', *Innovations in Systems and Software Engineering (ISSE)* **4**(3), 215–222.

Note

[1]http://www.systemc.org/
[2]www.papyrusuml.org/
[3]www.eclipse.org/emf/
[4]http://www.acceleo.org/pages/home/en
[5]http://www.omg.org/cgi-bin/doc?ad/2004-4-7
[6]http://www.model.com/
[7]http://www.xilinx.com/univ/xupv2p.html

MDA Based, SystemC Code Generation, Applied to Intensive Signal Processing Applications

Mickaël SAMYN, Samy MEFTALI, Jean-Luc DEKEYSER
Laboratoire d'Informatique Fondamentale de Lille
Université de Lille, France
http://www.lifl.fr

Abstract

In this paper, we presente a novel methodology for designing ISP applications specific SoCs. We generate in a systematic and automatic way, using the MDA approach, a SystemC specification at a timed transactional level.

The generated code describes the application mapped on a multiprocessor architecture. In addition to processors, this later can integrate several memories and a generic configurable interconnect.

The effectiveness of our methodology has been shown on an example of intensive signal processing application.

1 Introduction

Nowadays, SoC (System on Chip) design covers several domains and competences as : application modeling by aggregation of functional components, assembly of existing physical components, verification and the simulation of the modeled system, synthesis of a complete end-product integrated into a single chip.

Thus, modern complex SoCs include programmable processors, memory units (data/instructions), interconnection mechanisms and hardware functional units (Digital Signal Processors application specific circuits). These components can be generated for a particular application; they can also be obtained from IP (Intellectual Property) providers.

The ability to re-use software or hardware components is without any doubt a major asset for any codesign flow. But, unfortunately, usually IP transactions between providers and consumers take a very long time when the time to market constraints becomes more and more strong. So, generally SoC designers choose always to design describe some parts of the SoC, locally, by their development teams. This specification could be done using an HDL dependently on the needed abstraction level and also the nature of the component to design.

These last years have been marked by the apparence of SystemC [ABB+03], [SVD+02]. It is a system level description language, which becomes more and more used for SoC design. In fact, SystemC seems to be a good solution for SoCs specification at all the most used abstraction levels in the design. However, when some complex SoCs can integrate more than 200 million gates, it becomes very tedious and error prone to describe and to verify them, for any design team.

We believe that the Model Driven Architecture (MDA)[DBDM03] can enable us to propose a new method of system design respecting these principles. Indeed, it is based on the common UML modeling language to model all kinds of artifacts. The clear separation between the models and the platforms makes it easy to switch to a new technology while re-using the old designs. This may even be done automatically provided the right tools.

The MDA is an OMG proposed approach for system development. It primaly focuses on software development, but can be applied to any system development. The MDA is based on models describing the systems to be built. A system description is made of numerous models, each model representing a different level of abstraction. The modeled system can be deployed on one or more platforms via model to model transformations.

Thus, we present in this paper an MDA approach for SoCs design with SystemC as an output target language. The SystemC code is entirely automatically generated, so that to reduce as much as possible the specification errors and the time-to-market too. In order to make our tool realistic and usable to design complex SoCs, we choose to target exclusively intensive signal processing applications.

This application domain is composed of: systematic signal processing and intensive data processing. Many signal and image processing applications follow this organization: software radio receiver, sonar beam forming, or JPEG 2000 encoder/decoder. The systematic signal processing is the very first part of a signal processing application. It mainly consists of a chain of filters and regular processing applied on the input signals independently of the signal values. It results in a characterization of the input signals with values of interest. The intensive data processing is the second part a of a signal processing application. It applies irregular computations on the values issued by the systematic signal processing. Those computations may depend on signal values.

The remaining of this paper in organized as follow. Some significant works on high level design and our contribution are presented in section 2. Section 3 presents the three models that we use in our design flow: application, architecture and mapping models. We present in the section 4 the concepts that we defined in our SystemC meta model. The different steps of our methodology are illustrated on an example in the section 5. The section 6 concludes this paper.

2 Related works

We can find very few works in the literature that starts the system design from a very high abstraction level (UML for example) and generates automatically the SoC specification in a system level description language (SystemC) for a given class of applications.

One of the most significant works that we seen is SLOOP [ZMK+02] (System Level design with Object-Oriented Process). It is an UML2 approach for hardware/software codesign, based on four main models. The first one is called *conceptual model* and it describes the clients requirements. Above, there are two other models: *the functional model* which describes the application and the communication between processes; *the architecture model* describes the hardware resources. And finally, after the mapping phase, *the performance model*is used to see whatever the obtained performances are suitable with the *conceptual model*. SLOOP's approach seems to be complete and coherent, but with the used models data parallelism can not be expressed, and the hierarchy can not be modeled. This is unfortunately a big restriction for intensive signal processing specific SoC design.

MILAN [AAJ+01], [VK01] is an academic tool of the University of Southern California. It is a model based extensible framework that facilitates rapid, multi-granular performance evaluation of a large class of embedded systems, by seamlessly integrating different widely used simulators into a unified environment. MILAN provides a formal paradigm for specification of structural and behavioral aspects of embedded systems, an integrated model-based approach [MP02], and a unified software environment for system design and simulation. In this environment, the application is modeled, at a high abstraction level, using a visual editor looking like Ptolemy 2. The architecture is also modeled in the same way, then a SystemC specification can be generated. The main disadvantage of MILAN is that the application meta model is composed mainly by some very low level components. This makes the modeling of an industrial size ISP application

very tedious, and also limits the flexibility of the generated SystemC model.

2.1 Contribution

The main contribution of this paper is to present a completely automatic SystemC code generation tool for ISP application (PSM), starting from a high level model of a mapped application. In our tool, all the main concepts of ISP modeling are implemented in the SystemC metamodel. In addition to this, we target flexible and modular architectures and keep our metamodel flexible.

3 Models

The methodology and tool that we present in the remaining sections of this paper are integrated in a global project called Gaspard[1] [BDD+03]. This later permits to design ISP applications specific SoCs using the MDA approach. Our design tool starts from a PIM application and architecture specification, and then a PSM (SystemC) description of the SoC is generated. Thus we distinguish mainly three models in our tool: application, architecture and mapping models.

3.1 Application model : ISP

To model our applications, we use Array-OL [DLB+95], developed by TMS[2]. It is a programming language dedicated to signal processing applications. This application domain is characterized by systematic, regular and massively data-parallel computations. Array-OL applications are edited in a graphical environment, and are built on two levels: global level describing the application through a directed graph and a local level which details the computations performed on arrays elements by each node of the application. This later expresses dependencies between arrays elements.

3.1.1 Global model

The global model is an oriented acyclic graph composed by: nodes of arrays or tasks, connected by edges showing the execution order. Each task consumes at least one array, performs some computations and produces at least one array. The number of dimensions and arrays sizes are not bounded, but usually we use only one infinite dimension.

3.1.2 Local model

In the local stage, data computations are detailed. In Array-OL, arrays consumed and produced by a task are called pattern. A pattern can be an entire array or a sub-array extracted from another one. To extract a pattern, Array-OL introduces two matrixes, used to define the extraction of a pattern from an array, called paving and fitting. In fact, the Paving matrix expresses the origin of the current pattern and the fitting matrix expresses the current element of the pattern. For computations, another notion is needed in order to simulate the Array-OL application, it is the iteration domain. Thus, we have two vectors, paving vector and fitting vector. Using the paving vector, we can define the origin of the current pattern and with a complete iteration on fitting vector, all pattern data are covered. To create a pattern, the notion of notion is introduced. It is a task that puts (gets) some data to (from) an array, in a given order.

In the figure 1, there is one task that consumes two arrays, and produces one. Tilers are not represented, but their works was represented by arrows. They extract patterns from input

[1] Graphical Array Specification for Parallel and Distributed Computing
[2] Thomson Marconni Sonar

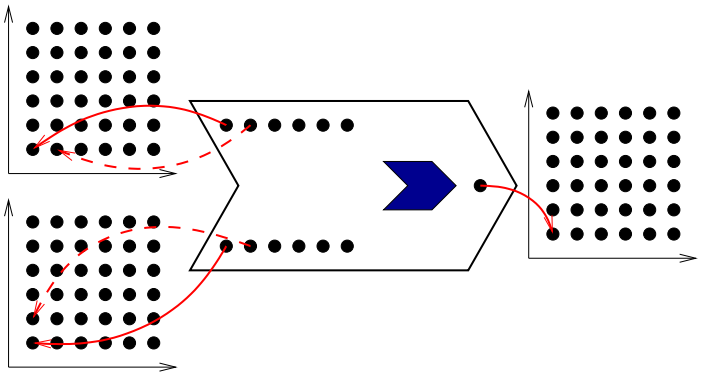


Figure 1: Example : Global task with one task and tree tiler inside

arrays to be consumed by the task which will produce then one pattern to be stored in the output array.

3.2 Architecture model

All modern intensive signal processing applications need more than one embedded processor, and so a multiprocessor platform in order to reach high performances. Thus, our SystemC meta model permits to generate PSM (Platform Specific Model) multiprocessor SoCs descriptions. This latter is mainly composed by three types of hardware components: (execution units) processors, memories and interconnects. These components are generated as SystemC modules described at a timed transactional level (PV-T). We choose this abstraction level for the code generation in order never have a large abstraction gape between the PIM (Platform Independent Model) mapped application model and the level of the generated code. This keeps so our code generation systematic and entirely automatic. The feed back to modify the mapped application are also faster and easier.

A typical SoC architecture that we are able to generate is illustrated in the figure 2. Several processors modules and memories can be connected to a generic configurable interconnect. This later is a kind of a high level NoC (Network on Chip). The choice of this type of interconnects is motivated by its high configurability are its very good cost/performances rapport at the implementation phase. In fact, many parameters of our generic interconnect as: the communication fifos sizes, saturation threshold, ..etc, can be configured automatically. Thus the system designer is able to fix all this parameters after a certain number of simulations and performances checks, so that the final NoC will be really specific for the application.

3.2.1 Note.

In addition to the configurable NoC, buses are also implemented in our SystemC meta model.

3.3 Mapped Model

3.3.1 Functional Verification

Before choosing a complex multiprocessor architecture for a given application, we first verify its functionality on a single processor architecture. Thus, to validate the ISP application, this later is mapped on single processor architecture (one processor, one memory). With this mapping, all modules are executed sequentially. The results of this execution show if the functionality of the application is valid and whatever it gives the right outputs for a set of input vectors.

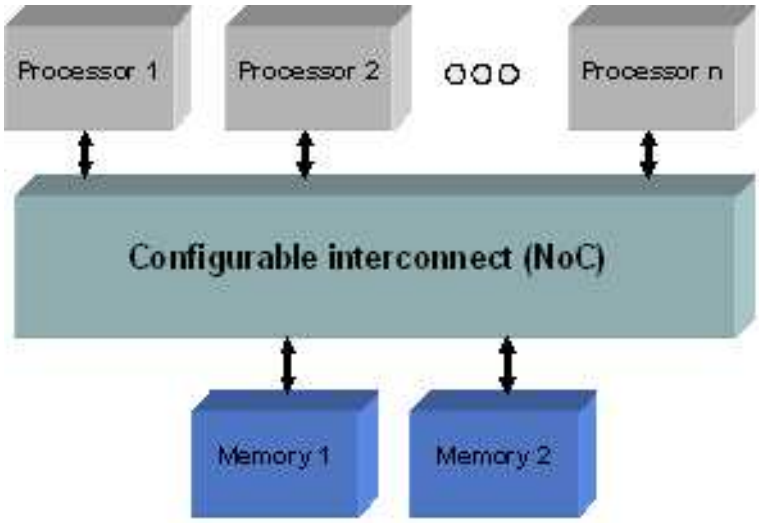


Figure 2: A typical multiprocessor architecture based on a configurable NoC

3.3.2 Mapping

Multiprocessor architectures platforms can be modeled using our architecture metamodel, and the operating mode has to be defined for the application mapping. Two operating modes can be outlined: pipeline mode and SPMD[3] mode. This mode can be applied to the whole application as well as locally (for a part of the application).

pipeline In pipeline mode, tilers and tasks are mapped on the processor. Thus, we distinguish two different cases. If two consecutive modules are on the same processor, communication is done internally, but if they are not on the same processor, they have to send an external signal, that we call synchronization signal, to communicate.

SPMD In SPMD mode, we make an instance per processor for each module. Each processor computes an independent part of the data. This is possible using the origin and splitting the iteration domain to suit to the data placement.

Synchronization As said before, synchronization is needed to synchronize two or more tasks mapped on different processors. There are two kind of synchronization: local and global. Local synchronization is used to synchronize two or more modules, to have the correct scheduling. Global synchronization is used to synchronize all processors (useful in SPMD mode), this kind of synchronization is called synchronization barrier.

4 SystemC

In this section, developed SystemC modules for the code generation are presented. Then, operating modes between these modules are discussed to show how the system works. We distinguish manly two kind of modules : active and passive.

[3]Single Program, Multiple Data

4.1 Active

Active components are all modules that compute or initiate communication with the others. We defined two types of active modules, modules to be designed (tiler, task, processor) and modules that are generated for a simulation purpose (MemoryAccess and Synchronization).

4.1.1 Processor

A processor is a SystemC module that contains one or more tasks. It is hierarchical and able to integrate tiler modules. Processor is an interface between interconnect media and other modules. It has various ports : five ports used for memory control, one port to start processing, one port to notify the end of a process execution and one port per processor in the architecture for synchronization purpose. A processor contains also one MemoryAccess and one Synchronization module.

4.1.2 MemoryAccess

The MemoryAccess module has been developed to simplify tiler and task development extracting memory management. It has three operating mode.

- **Block transfer** In this mode, the module transfers data from the input block to the output one. This mode is used by tilers.
- **Read data** In this mode, the module reads data from the memory and transfers it to a processor's task.
- **Write data** In this mode, the module writes data in the memory. This data comes form a processor's task.

The MemoryAccess has two address vector ports: one for the input block and one for the output. It also has five memory control ports, directly connected to the processors ones. The module has one port for task's data. And, finally, two control ports for the start and the stop signals.

To understand what a MemoryAccess do, let see an extract of the code. This extract represents the port declaration for this module.

```
1   sc_port<sc_signal_in_if<bool>, 0> valideMA ;
2   sc_port<sc_signal_inout_if<bool>, 0> finish ;
3   sc_port<sc_signal_inout_if<myVector<int>*>, 0> pattern ;
4   sc_port<sc_signal_in_if<myVector<address>*>, 0> addressListSource ;
5   sc_port<sc_signal_in_if<myVector<address>*>, 0> addressListDestination ;
6   sc_port<sc_signal_in_if<bool>, 0> functionSelect;
7   sc_port<sc_signal_inout_if<address>, 0> memoryAddress ;
8   sc_port<sc_signal_inout_if<bool>, 0> selectMode ;
9   sc_port<sc_signal_inout_if<bool>, 0> valideMemory ;
10  sc_port<sc_signal_inout_if<int>, 0> dataItem ;
11  sc_port<sc_signal_in_if<bool>, 0> ackMem ;
```

In lines 1 and 2, two ports are declared, one to start the MemoryAccess computation and the other to notify the end of the current computation. In the line 3, one port to communicate with a task is declared. This port is used to get or put a pattern in the memory. In lines 4 and 5, the two declared ports are used in various ways. It depends on the port's value. In fact, in

line 6, if the value is set to true, they are used to move a data block form Source to Destination, and if the value is set to false, there will be two possibilities. If there is a value on the Source port, MemoryAccess extracts the pattern associated to this value and put it on the pattern port. It there is a value on the Destination port, the MemoryAccess put the pattern present in the pattern port to memory. The others ports in lines 7 to 11, are used to control the memory.

4.1.3 Synchronization

Used for the simulation of multiprocessor platforms, this modules drives the execution. At the code generation step, a synchronization table is created and for each task or tiler, an entry is created. This entry contains the number of synchronization signals per processor that task/tiler need to be executed.

4.1.4 Task

A task module is a SystemC container for a C++ function. This function takes arrays in input and produces other arrays as output. This SystemC module is used for scheduling and synchronization purpose. It also access the memory through the MemoryAccess module.

4.1.5 Tiler

A tiler is a module that only computes addresses to create a pattern or to put it in a memory. In the first case, Tiler module computes addresses to get data from the input array, and puts this data in a specific memory location, to be used by a task or an other tiler. In the second case, Tiler computes addresses to move data from a pattern to the memory location corresponding at the computed address.

4.2 Passive

Passive components are modules communicating only by responding to a previous query (initiated by an active module). All communication medium can be assimilated to this component type because they do not initiate any communication.

4.2.1 Interconnect

An interconnect is a SystemC module used to allow communication between connected module. An interconnect may be a simple bus, a crossbar, or anything else that allows inter modules communication.

4.2.2 Memory

A Memory is a module used to simulate real memory operations. There are several kinds of memories. Thus, in order to not be restrictive, we choose to use multi ports, multi bancs memories.

4.3 Synchronization Operating mode

After receiving a start signal from the initiator module, the synchronization module sends a start signal to the target module if all synchronization required (from all needed initiators) have been received. The synchronization signal between processors is a record of two fields: processor

modules name and initiator modules name. This permits to the synchronization module of a given processor to select just needed signals.

5 Application Example

To illustrate our methodology and to show its efficiency, we modeled an intensive signal processing application and a multiprocessor architecture in UML 2.0. The mapping, scheduling and the SystemC code generation have been then performed automatically. These steps are presented with details in the remaining of this section.

5.1 Application

This application is composed by four tasks: T1, T2, T3 and T4, and three tillers: ti1, ti2 and ti3. Each one of the tasks T1, T2 and T4 consumes an array and produces an other. The task T3 consumes 2 arrays. It is a hierarchical task and contains three sub-task. We choose to represent this application as shown in the figure 3 and not directly in the UML representation for clarity reasons.

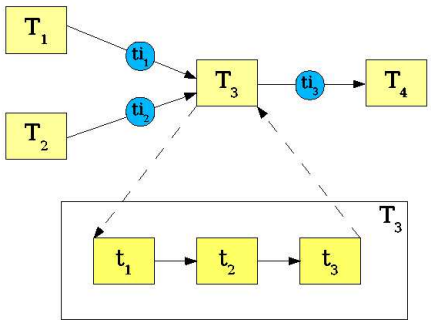


Figure 3: Application example

5.2 Architecture

The architecture that we choose for this application is composed by three processors (CPU), one global shared memory and a bus. This architecture is illustrated in the figure 4

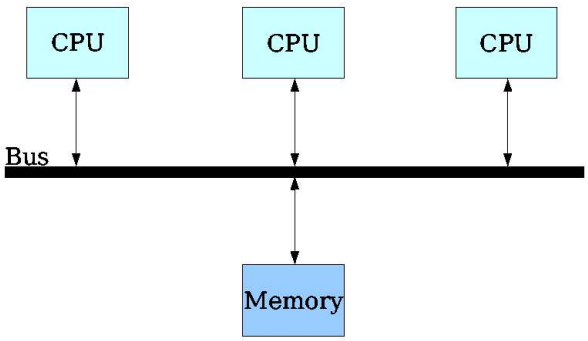


Figure 4: Architecture example

5.3 Mapping and Scheduling

After application and architecture modeling, the next step in our design flow is the mapping. It consists in distributing all the tasks on the execution units of the architecture (CPUs). In our example, tasks T_1 and T_2 can be executed simultaneously as there are no dependencies between them. Tilers ti_1 and ti_2, task T_3 (sub-tasks: t_1, t_2, t_3) and ti_3, have to be executed sequentially until the end T_3 consumes data furnished by t_1 and t_2 at each iteration, and then the tiler ti_3 consumes the data returned by T_3. T_4 will be the last executed task. This execution scheduling will be repeated until the end of the iteration space. In order to illustrate several execution modes, we decided to map T_1, T_2 and T_4 in pipeline mode tasks, and T_3 with all tilers in SPMD mode. The scheduling obtained with this mapping is shown in the figure 5).

Note. In this mapping, splitten the iteration space in three part, to map this application part on the three processor available.

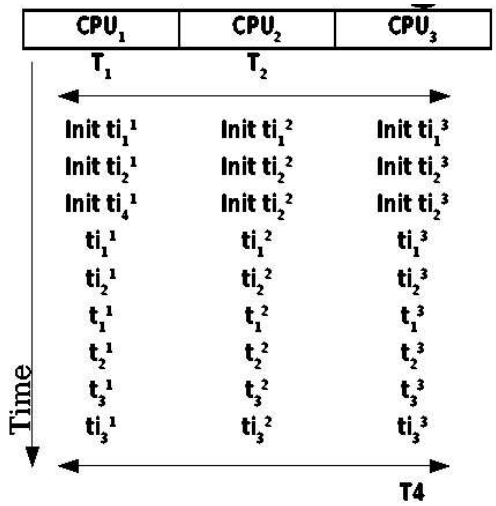


Figure 5: Scheduling

The figure 6shows the associated mapping diagram to the application. This will be used to create the SystemC model permitting SystemC code generation. In this diagram, we represented only scheduling dependencies. All memory accesses are done by the MemoryAccess module. Thus, we can see all modules mapped on the corresponding processors. We note in this figure that all inter modules synchronization signals are transmitted first by the sender module to the synchronization module, then by this latter to the bus.

5.4 Generated SystemC code

The SystemC code corresponding to mapped application has been automatically generated. Some characteristics of the generated SystemC code are summarized in the following table.

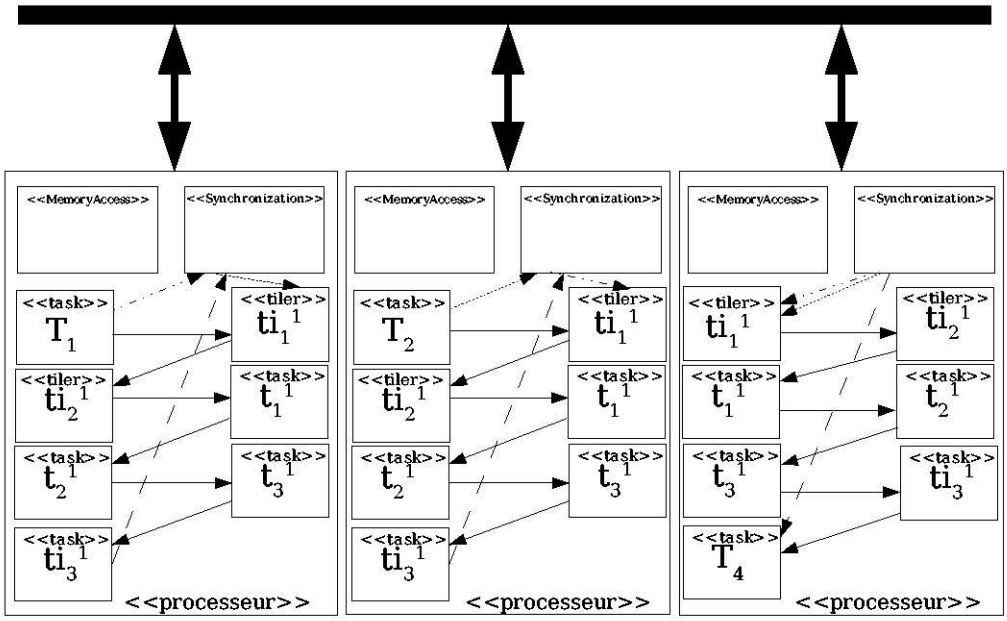


Figure 6: Mapping

	File Information			
	Number of file	Line/file	Total line	Proportion of configured code (%)
Processor	3	700	2100	95
Task	12	200	2400	10*
Tiler	9	500	4500	10
Bus	1	500	500	80
Memory	1	100	100	5
Total	26	351.92	9150	40

* Depend on the used function

We can see that our application basics blocks (tiler and task) are mainly constituted by configurable code. This is very natural because: a tiler is a generic module not very dependent on the application, so for its generation we just have to give all initialization values; for a task, the rate of the configured code depends on the function used inside. Memory generation is very similar to the tiler one, it only need some initialization value to work. But we generate a great part of the code describing processors and the bus modules. In fact, we have to generate all module instantiations in the processor and to generate all connections between this module and the synchronization table for inter-processor communication. For the bus, we generate all ports and communication protocols.

6 Conclusion and future work

In this paper, we presented a novel methodology for designing ISP applications specific SoCs. We generate in a systematic and automatic way, using the MDA approach, a SystemC specification at a timed transactional level.

The generated code describes the application mapped on a multiprocessor architecture. In addition to processors, this later can integrate several memories and a generic configurable

interconnect.

The effectiveness of our methodology has been shown on an example.

We are bow working on a second version of our tool. In this latter, the generated SystemC modules will be compliant with the OCP interface standards. This will permits to designers the use of several SystemC IPs available in the emerging open sources IP libraries like SoCLib.

References

[AAJ+01] Agrawal A, Bakshi A, Davis J, Eames B, Ledeczi A, Mohanty S, Mathur V, Neema S, Nordstrom G, Prasanna V, Raghavendra C, and Singh M. "milan: A model based integrated simulation framework for design of embedded systems". In *"Workshop on Languages, Compilers, and Tools for Embedded Systems"*, Snowbird, Utah, June 2001.

[ABB+03] EL Mustapha Aboulhamid, Mike Baird, Bishnupriya Bhattacharya, David Black, Dundar Dumlogal, Abhijit Ghoshand Andy Goodrich, Robert Graulich, Thorsten Groetker, Martin Jannsenand, Evan Lavelle, Kevin Kranen, Wolfgang Mueller, Kurt Schwartzand Adam Rose, Ray Ryan, Minoru Shoji, and Stuart Swan. *SystemC 2.0.1 Language Reference Manual.* 2003.

[BDD+03] Pierre Boulet, Jean-Luc Dekeyser, Cédric Dumoulin, Philippe Marquet, Philippe Kajfasz, and Dominique Ragot. Sophocles: Cyber-enterprise for system-on-chip distributed simulation – model unification. In *IFIP International Workshop On IP Based System-on-Chip Design*, pages 325–330, November 2003.

[DBDM03] Cédric Dumoulin, Pierre Boulet, Jean-Luc Dekeyser, and Philippe Marquet. MDA for SoC design, intensive signal processing experiment. In *FDL'03*, Frankfurt am Main, September 2003. ECSI.

[DLB+95] Alain Demeure, Anne Lafage, Emmanuel Boutillon, Didier Rozzonelli, Jean-Claude Dufourd, and Jean-Louis Marro. Array-OL : Proposition d'un formalisme tableau pour le traitement de signal multi-dimensionnel. In *Gretsi*, Juan-Les-Pins, France, September 1995.

[MP02] Sumit Mohanty and Viktor K. Prasanna. "rapid system-level performance evaluation and optimization for application mapping onto soc architectures". In *"15th IEEE International ASIC/SOC Conference"*, New York, USA, 2002.

[SVD+02] Stuart Swan, Dirk Vermeersch, Dündar Dumlugöl, Peter Hardee, Takashi Hasegawa, Adam Rose, Marcello Coppola, Martin Janssen, Thorsten Grötker, Abhijit Ghosh, and Kevin Kranen. *Functional specification for systemc 2.0.* 2002.

[VK01] Mathur V. and Prasanna V. K. "a hierarchical simulation framework for application development on system-on-chip architectures". In *"14th IEEE Int'l ASIC-SOC Conference"*, Washington DC, USA, September 2001.

[ZMK+02] Qiang Zhu, Akio Matsuda, Shinya Kuwamura, Tsuneo Nakata, and Minoru Shoji. An obect-oriented design process for system-on-chip using UML. In *Proceedings of the 15th international symposium on System Synthesis*, pages 249–259, Kyoto, Japan, 2002.