UNIVERSITE JOSEPH FOURRIER – GRENOBLE1

SCIENCE & MEDICINE

THESE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER

Spécialité : Microélectronique

Présentée et soutenue publiquement

Par

Arif SASONGKO

Le 15 Octobre 2004

Titre

Prototypage basé sur une plateforme reconfigurable pour la vérification des systèmes monopuces

Directeur de thèse : Ahmed Amine JERRAYA

JURY

Marc RENAUDIN	Président
Fabrice KORDON	Rapporteur
Michel ROBERT	Rapporteur
Ahmed A. JERRAYA	Directeur de thèse
Frédéric ROUSSEAU	Co-directeur de thèse

Thèse préparée au sein du laboratoire Technique de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs - TIMA

SOMMAIRE

Sommaire i

Liste des figures v

Liste des tableaux vii

Chapitre 1 : Introduction 1

1.1 Contexte : conception des systèmes multiprocesseurs monopuces 2
1.1.1 Introduction 2
1.1.2 La conceptions des systèmes monopuces 3

1.2 Nécessité de la vérification 5

1.3 Les techniques de vérification des systèmes monopuces 6

1.4 Nécessité du prototypage pour réduire le temps de conception 7

1.5 Contributions 9
1.5.1 Définition d'un flot de prototypage basé sur une plateforme reconfigurable 9
1.5.2 Automatisation du flot du prototypage 9
1.5.3 Proposition d'une solution de co-émulation basée sur une plateforme reconfigurable 9

1.6 Plan du mémoire 10

Chapitre 2 : Vérification des systèmes monopuces : techniques et comparaison 11

2.1 Introduction 12

2.2 Objectif de la vérification d'un système monopuce 12
2.2.1 Vérifier la spécification fonctionnelle 13
2.2.2 Vérifier l'architecture du système 13
2.2.3 Vérifier l'implémentation des composants du système 13
2.2.4 Vérifier l'intégration de composants hétérogènes 14
2.2.5 Détecter tous les bogues avant la fabrication 14

2.3 Difficultés de la vérification d'un système monopuce 15
2.3.1 Hétérogénéité 15
2.3.2 Complexité 15
2.3.3 Il n'existe pas encore d'outils de déboguage de systèmes multiprocesseurs 16

2.4 Diverses techniques de vérification des systèmes monopuces 16
2.4.1 Critères de comparaison des techniques de vérification des systèmes monopuces 16
2.4.2 La vérification formelle 18
2.4.3 La simulation sur ordinateur 20
2.4.4 L'émulation 23
2.4.5 Le prototypage 25
2.4.6 Synthèse des différentes techniques 28

2.5 Le prototypage basé sur une plateforme reconfigurable 30
2.5.1 Intérêt du prototypage pour les systèmes monopuces 30
2.5.2 Intérêt du prototypage basé sur une plateforme reconfigurable 30
2.5.3 Intérêt de la co-émulation basée sur une plateforme reconfigurable 31
2.5.4 Exemples de flot de prototypage industrielle 31

2.6 Conclusion **37**

Chapitre 3 : Définition d'un flot de prototypage sur une plateforme reconfigurable 39

3.1 Introduction **40**

3.2 Définition du prototypage d'un système monopuce sur une plateforme reconfigurable **40**
3.2.1 Modèle de système monopuce 40
3.2.2 Modèle générique d'une plateforme de prototypage 42
3.2.3 Modèle d'une plateforme idéale pour le prototypage 43

3.3 Les plateformes reconfigurables et les flots de prototypage existants **46**
3.3.1 La classification des plateformes selon l'utilisation 46
3.3.2 La classification de la plateforme reconfigurables par le nombre de domaine d'application supporté par la plateforme 46
3.3.3 La classification des plateformes reconfigurables selon la configuration ou les types de composants configurables dans la plateforme 47

3.4 Flot de prototypage simple **48**
3.4.1 Allocation 48
3.4.2 Génération de code 49
3.4.3 Exemple 49

3.5 Flot de prototypage proposé **51**
3.5.1 Configuration 52
3.5.2 Adaptation 54

3.6 Conclusion **56**

Chapitre 4 : Applications 57

4.1 Introduction **58**

4.2 Plateforme ARM Integrator **58**
4.2.1 La carte mère de plateforme ARM Integrator 59
4.2.2 La carte processeur de la plateforme ARM Integrator 60
4.2.3 La carte FPGA de la plateforme ARM Integrator 60
4.2.4 Le Bus AMBA 61
4.2.5 Outil de déboguage : Multi-ICE 61
4.2.6 Logiciel de développement et de déboguage 62
4.2.7 Les caractéristiques de la plateforme ARM Integrator 63

4.3 Prototypage du VDSL sur la plateforme avec un réseau de communication fixé **64**
4.3.1 Description de l'application 64
4.3.2 Spécification de l'application 64
4.3.3 Architecture logicielle matérielle 65
4.3.4 Objectif du prototypage et contraintes 65
4.3.5 Allocation 66
4.3.6 Configuration 67
4.3.7 Adaptation 69
4.3.8 Génération de code 69
4.3.9 Résultats 70

4.4 Prototypage du DivX **71**
4.4.1 Description de l'application 71
4.4.2 Objectif du prototypage et contraintes 72
4.4.3 La spécification et l'architecture RTL 73
4.4.4 Allocation 76
4.4.5 Configuration 77
4.4.6 Adaptation 77
4.4.7 Génération du code 81
4.4.8 Résultats et analyse 81

Chapitre 5 : Analyse et perspectives....83

5.1 Introduction....84

5.2 Evaluation des exemples....84
5.2.1 VDSL....84
5.2.2 DivX....85
5.2.3 Synthèse....86

5.3 Automatisation de l'adaptation pour accélérer le développement du logiciel....87
5.3.1 Les outils de génération automatique du flot ROSES....87
5.3.2 L'automatisation de l'adaptation....93
5.3.3 Avantages et limitations....96
5.3.4 Conclusion sur l'adaptation....98

5.4 Co-émulation....98
5.4.1 Exemple de co-émulation en utilisant la plateforme ARM Integrator....98
5.4.2 Evaluation....104
5.4.3 Perspectives....105

Chapitre 6 : Conclusion....107

Bibliographie....111

Publications....117

LISTE DES FIGURES

Figure 1 Représentation de système monopuce 3
Figure 2 Flot de conception des systèmes monopuces 4
Figure 3 Effet du prototypage sur la conception d'un système monopuce 8
Figure 4 Création d'un produit numérique 12
Figure 5 Le System Explorer d'Aptix 32
Figure 6 Les phases dans la conception des système monopuces d'Aptix 33
Figure 7 Flot de prototypage d'APTIX 34
Figure 8 La plateforme de prototypage FlexBench 35
Figure 9 La topologie de FlexBench 36
Figure 10 Flot de prototypage FlexBench 37
Figure 11 Modèle d'architecture d'un système monopuce 41
Figure 12 Modèle générique de plateforme de prototypage 42
Figure 13 L'implémentation d'une plateforme idéale 44
Figure 14 modèle de notre implémentation de la plateforme idéale 45
Figure 15 Flot de prototypage simple 48
Figure 16 Exemple de l'allocation 49
Figure 17 Génération du code 51
Figure 18 Flot de prototypage 52
Figure 19 Exemple de configuration 53
Figure 20 Configuration 54
Figure 21 Adaptation 55
Figure 22 Plateforme ARM Integrator avec 3 cartes processeur et 2 cartes FPGA 58
Figure 23 La carte mère 59
Figure 24 La carte processeur 60
Figure 25 La carte FPGA 61
Figure 26 La configuration typique d'une bus AMBA 61
Figure 27 Multi-ICE 62
Figure 28 Modem VDSL 65
Figure 29 Architecture logiciel/matériel de modem VDSL au niveau RTL 66
Figure 30 Configuration de l'application VDSL 68

Figure 31 Convertisseur 69
Figure 32 La génération de code 70
Figure 33 Schème de DivX encodeur 71
Figure 34 La spécification de départ de l'application DivX 73
Figure 35 L'architecture logicielle/matériel de l'application DivX 74
Figure 36 Décomposition des donnée 74
Figure 37 Implémentation de la partie communication de l'exemple DivX 75
Figure 38 Allocation de DivX 76
Figure 39 Revenir à la spécification plus haut niveau 78
Figure 40 Génération d'interface logicielle 78
Figure 41 Implémentation de la partie communication sur le prototype de l'application DivX 79
Figure 42 Interface logiciel/matériel 88
Figure 43 Le flot ROSES 89
Figure 44 Un exemple de modules, de liens, et de ports 90
Figure 45 Un exemple d'entée du flot ROSES 91
Figure 46 ASOG 92
Figure 47 Génération des interfaces matérielles par ASAG 93
Figure 48 Abstraction et modification de paramètres 94
Figure 49 Génération de la couche d'abstraction du matériel 95
Figure 50 L'adaptation pour développer le logiciel 97
Figure 51 Co-simulation avec SystemC 99
Figure 52 Connexion entre une simulation SystemC et un émulateur 99
Figure 53 Co-émulation en utilisant une connexion multi-ICE 100
Figure 54 Co-émulation en utilisant une carte réseau 100
Figure 55 Co-émulation en utilisant un pont PCI/PCI 101
Figure 56 Co-émulation en utilisant une connexion USB 102
Figure 57 L'expérimentation sur co-émulation 103

LISTE DES TABLEAUX

Tableau 1 Evolution de la technologie d’intégration sur silicium [Cha99] 7
Tableau 2 Tableau récapitulatif de technique de vérification.....29
Tableau 3 Allocation de l'architecture du VDSL sur la plateforme ARM Integrator 67

CHAPITRE 1 : INTRODUCTION

1.1 Contexte : conception des systèmes multiprocesseurs monopuces....2
1.1.1 Introduction....2
1.1.2 La conceptions des systèmes monopuces....3
1.2 Nécessité de la vérification....5
1.3 Les techniques de vérification des systèmes monopuces....6
1.4 Nécessité du prototypage pour réduire le temps de conception....7
1.5 Contributions....9
1.5.1 Définition d'un flot de prototypage basé sur une plateforme reconfigurable....9
1.5.2 Automatisation du flot du prototypage....9
1.5.3 Proposition d'une solution de co-émulation basée sur une plateforme reconfigurable....9
1.6 Plan du mémoire....10

1.1 Contexte : conception des systèmes multiprocesseurs monopuces

1.1.1 Introduction

Cette thèse s'inscrit dans le contexte de la conception des systèmes monopuces. Ces systèmes intègrent sur une même puce plusieurs processeurs ainsi que des composants complexes et hétérogènes (mémoires, périphériques, unités de calcul spécifiques). On utilise le terme « Multiprocessor System-on-Chip/MPSoC » en anglais pour décrire ces systèmes.

Ces systèmes prennent une place de plus en plus importante dans l'industrie de la microélectronique. Les prévisions d'ITRS prévoient qu'en 2005, 70 % des ASIC intégreront au moins un processeur. Cela se confirme par l'existence de nombreuses applications complexes telles que les téléphones portables, les «set-top box», les consoles de jeux, les appareilles photos numériques qui intègrent des systèmes monopuces comprenant un ou plusieurs processeurs. Toutes ces applications correspondent à des productions de masse et requièrent des performances très élevées, ce à quoi répondent les systèmes monopuces. Ces systèmes semblent devenir les systèmes pilote de l'industrie microélectronique du futur.

Les systèmes monopuces sont généralement dédiés à des applications spécifiques. Comme les systèmes embarqués classiques, ils comportent des parties logicielles (programme s'exécutant sur un processeur, DSP, ou autre type de processeur) et des parties matérielles (composants spécifiques). Les fonctionnalités offertes par les parties logicielles et matérielles sont spécifiquement adaptées pour respecter les objectifs (coût, performance, consommation,…).

La Figure 1 présente un système monopuce avec des parties matérielles et des parties logicielles. Pour répondre aux contraints de temps de mise sur le marché et de coût, l'architecture matérielle d'un système multiprocesseur monopuce est obtenue par l'assemblage de composants connectés à travers un réseau de communication.

Le développement des parties logicielles devient tellement complexe, qu'il est nécessaire de le commencer le plus tôt possible, et ne plus attendre que le circuit (en fait les parties matérielles) soit disponibles. Le logiciel est organisé en couches. Chaque couche traite un aspect différent : abstraction du matériel, gestion des ressources, protocoles de communication, etc.

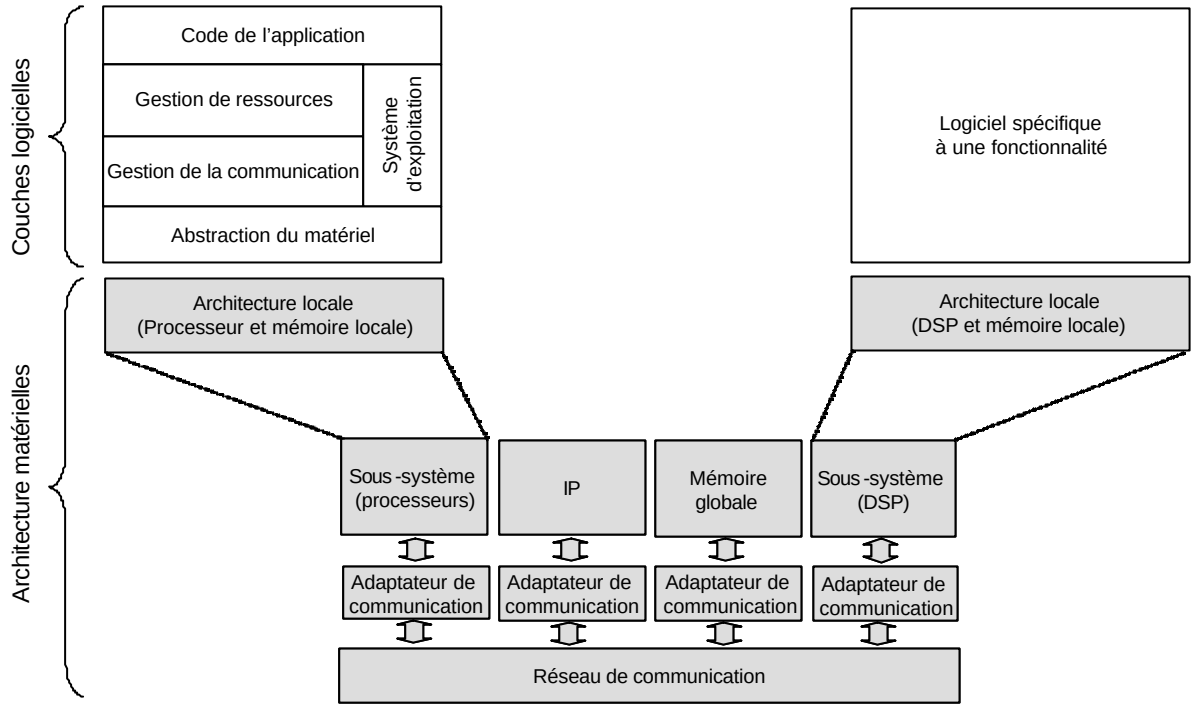


Figure 1 Représentation de système monopuce

1.1.2 La conceptions des systèmes monopuces

La complexité des systèmes multiprocesseurs monopuces et l'hétérogénéité des composants sont telles que les méthodes de conception d'ASIC ou de systèmes embarqués ne peuvent plus s'appliquer. Il a donc fallu trouver une nouvelle façon de concevoir ces systèmes. Leur conception se décompose en plusieurs étapes (Figure 2) et commence à un haut niveau d'abstraction, ce qui permet de s'affranchir de nombreux détails.

A un haut niveau d'abstraction, on s'intéresse à la fonctionnalité, indépendamment de l'implémentation finale, ce qui correspond à la conception de niveau système. A ce niveau de conception, on recherche des algorithmes et des représentations de données les plus adaptés pour réaliser la spécification. On obtient une spécification fonctionnelle que l'on valide généralement par simulation.

Quand la spécification fonctionnelle est correcte, l'étape suivante consiste à trouver une architecture efficace. C'est l'exploration d'architectures qui détermine l'implémentation logicielle ou matérielle de tous les composants. Grossièrement, les composants qui nécessitent des performances élevées sont implémentés en matériel, et ceux qui nécessitent de la flexibilité sont implémentés en logiciel. On choisit aussi dans cette étape les composants physiques (processeur, DSP), qui exécuteront les parties logicielles, ainsi que

l'architecture mémoire, la gestion des E/S, …. A la fin de cette étape, on obtient les spécifications de chaque composant matériel et logiciel.

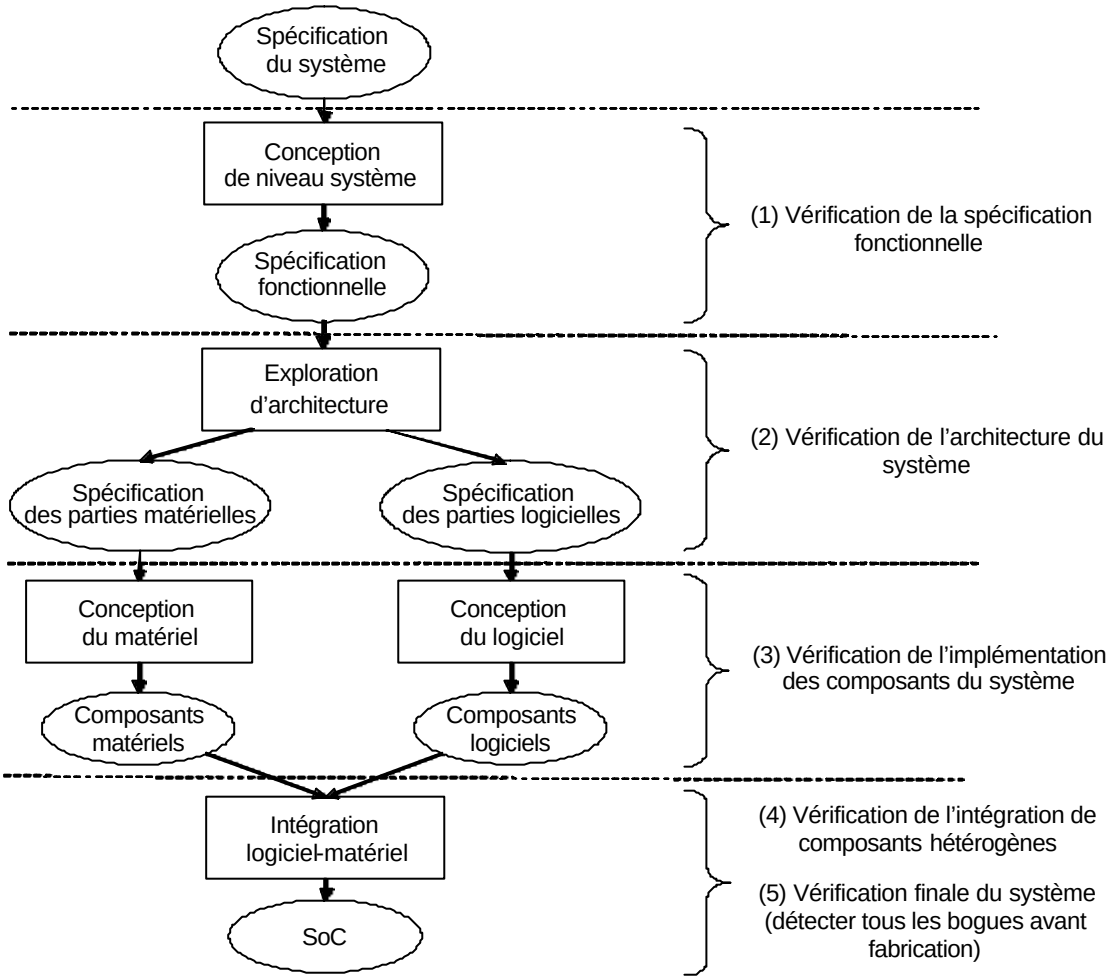


Figure 2 Flot de conception des systèmes monopuces

Les étapes suivantes sont la conception du matériel et du logiciel. Le principe est de réutiliser des composants existants, ce qui permet de gagner du temps par rapport à une conception complète de tout le système. Pour les composants matériels qui n'existent pas, la conception peut suivre le flot de conception traditionnel avec les différentes étapes de synthèse (comportementale, logique puis physique). Le logiciel est implémenté en couche pour séparer les différentes fonctionnalités. Au plus bas niveau, des pilotes ("Hardware Abstraction Layer") permettent d'accéder aux ressources matérielles. Au dessus, le système d'exploitation gère l'exécution des différentes taches de l'application, ainsi que les E/S. Enfin le code de l'application s'exécute à travers le système d'exploitation (Figure 1).

La dernière phase est la phase d'intégration logiciel/matériel. Il s'agit d'une part de vérifier que le logiciel s'exécute correctement sur les composants programmables, mais aussi que les échanges d'informations entre les composants sont corrects. Pour ces échanges d'informations, des composants d'interface (adaptateur de communication) sont nécessaires et sont placés entre les composants et le réseau de communication pour adapter les différents protocoles et le type de données. Entre un processeur et le réseau de communication, ces adaptateurs de communication peuvent être complexes avec une partie logicielle (pilotes du processeur) et une partie matérielle (composant d'interface). La réalisation de ces composants d'interface est une des difficultés de la conception de SoC, et leur génération automatique est un des axes de recherche de l'équipe SLS du laboratoire TIMA. Ceci facilite l'exploration d'architectures, accélère la conception, et réduit les erreurs lors de la conception de ces interfaces.

1.2 Nécessité de la vérification

A chaque étape de la conception, le concepteur doit s'assurer que les nouveaux composants ou les nouveaux détails de réalisation qu'il rajoute font que la nouvelle description du système respecte toujours les spécifications.

La vérification des systèmes monopuces est un processus exigeant. A cause de leur complexité et de leur hétérogénéité, il faut des techniques qui permettent de vérifier plusieurs aspects : aspects fonctionnel et architectural, aspects logiciel et matériel, et la communication. Naturellement, les concepteurs qui effectuent ces vérifications doivent posséder toutes ces compétences techniques.

La vérification de tels systèmes est une étape clé dans leur conception. La vérification s'applique autour d'au moins cinq points montrés sur la Figure 2 :

1. *Pour vérifier la fonctionnalité*
2. *Pour vérifier l'architecture du système*
3. *Pour vérifier les fonctionnalités et la compatibilité des composants hétérogènes*
4. *Pour vérifier l'intégration des composants.*
5. *Pour tester le système dans son environnement avant sa fabrication.*

Le coût de vérification des systèmes monopuces est très élevé surtout en terme d'effort et de temps. En effet, on estime que 70 % du temps de conceptions est passé pour effectuer la vérification.

Le coût financier de la conception est aussi très élevé. Le prix d'un jeu de masques dans une technologie avancée étant de l'ordre de 0,5 millions d'euros, une erreur détectée après la fabrication du circuit entraîne un surcoût. Pour cette raison, le système doit être rigoureusement vérifié avant sa fabrication, ce qui permet de réduire de façon significative son coût.

1.3 Les techniques de vérification des systèmes monopuces

Il existe plusieurs techniques de vérifications : la vérification formelle, la simulation, l'émulation, et le prototypage. Chacune de ces techniques utilise un modèle différent. La vérification formelle emploie des notations formelles pour représenter les caractéristiques du système qui permettent de vérifier certaines propriétés. La technique de simulation emploie des modèles d'exécution et des modèles de calcul sur ordinateur. L'émulation emploie des modèles physiques qui imitent le comportement du matériel. Et le prototypage combine ces divers modèles et composants matériels.

Chacune de ces techniques possède ses caractéristiques, ce qui leur donne plus ou moins d'intérêt selon ce qui est attendu de la vérification. Par exemple, la simulation est considérée comme d'une grande flexibilité, puisque changer le modèle d'une fonction ou d'un composant dans la représentation du système est très facile. C'est actuellement la technique la plus utilisée. Mais on atteint ses limites de performances dès qu'il s'agit d'une simulation à bas niveau d'abstraction (avec tous les détails de réalisation) en recherchant une précision élevée.

Le prototypage consiste à réaliser le système sur une architecture différente de celle prévue pour la réalisation finale. Tous les composants ou sous-systèmes de la description sont soit remplacés par des composants physiques (logiciel ou matériel), soit émulés ou simulés. Le développement d'un prototype spécifique à une application est très coûteux, et on préfère une plateforme « générique », permettant de réaliser plusieurs applications.

Un autre problème apparaît dans la vérification quand un composant physique n'est pas encore disponible. Ceci repousse la réalisation du prototype et augmente aussi le temps

de conception. Dans ce cas, une solution mixte basée sur le prototypage et la simulation (co-émulation) semble plus appropriée.

1.4 Nécessité du prototypage pour réduire le temps de conception

Malgré la complexité des applications, les réalités du marché exigent un temps de conception de plus en plus court. En effet, la compétition pour un marché de masse est exigeante et l'évolution des applications est très rapide. Le Tableau 1 donne une idée de l'évolution des applications, de leur coût et de leur temps de développement dans les 8 dernières années [CHA99]. On peut voir que la capacité d'intégration et la complexité de l'application ont augmentées rapidement. Mais le temps de conception des nouveaux produits et des produits dérivatifs est réduit.

Pour avoir un idée de l'importance du temps de conception et de détection des bogues avant fabrication, on peut revenir sur un exemple provenant de NEC [DAL00]. Pour fabriquer son circuit « set-top box », le coût des trente masques nécessaire était de 1,2 million dollars américain, en technologie 0,13 micron. A cette époque, NEC prévoyait un retard de deux mois dans le lancement de son circuit, ce qui devait lui coûter plus de 50 millions de dollars en terme de vente.

	1997	1998	1999	2002
technologie du process	0,35 µm	0,25 µm	0,18 µm	0,13 µm
cout d'instalation de l'usine (en dollar américain)	1,5-2,0 milliards	2,0-3,0 milliards	3.0-4,0 milliards	> 4,0 milliards
temps de conception	18 - 12 mois	12 - 10 mois	10 - 8 mois	8 - 6 mois
temps de conception d'un produit dérivé	8 - 6 mois	6 - 4 mois	4 - 2 mois	3 - 2 mois
capacitè d'intégration sur silicium	200 K - 500 K portes logiques	1 M - 2 M portes logiques	4 M - 6 M portes logiques	10 M - 25 M portes logiques
application typique	telephone portable, PDA, DVD	Set-top boxes, PDA sans fil	applications internet	contrôleur interconnecté et intelligent

Tableau 1 Evolution de la technologie d'intégration sur silicium [Cha99]

Les systèmes monopuces contiennent à la fois des composants matériels complexes (processeur, décodeurs vidéo, audio, etc.) et une énorme quantité de logiciel. On entend par logiciel le programme de l'application, mais aussi les systèmes d'exploitation et les pilotes d'E/S. Le logiciel devient tellement complexe que sa vérification n'est plus réaliste en

utilisant des approches classiques de simulation basée sur des ISS (simulateur de processeur en instruction près, Anglais : instruction set simulator). En effet, l'enquête de EDC découvre que 70 % des concepteurs de système embarqué dépense plus de 50% du temps de conception pour développer de la partie logiciel [EVA03].

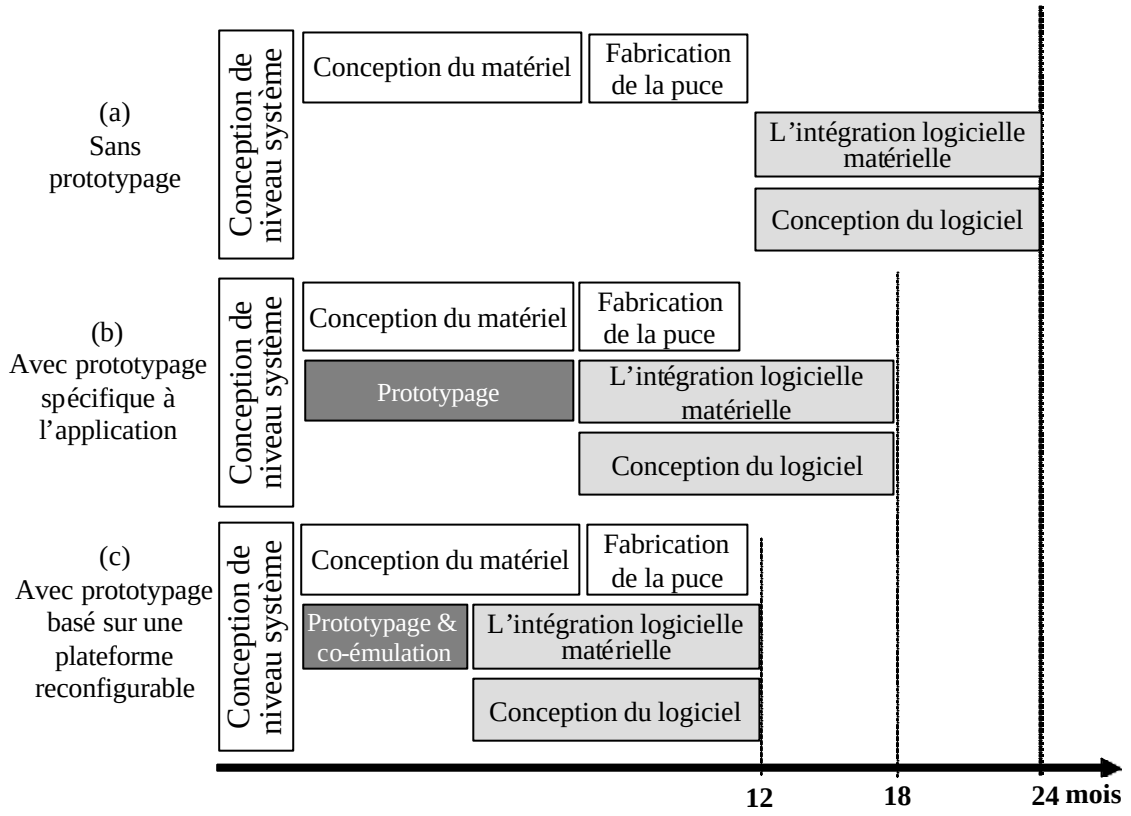


Figure 3 Effet du prototypage sur la conception d'un système monopuce

La Figure 3 montre l'intérêt du prototypage en raccourcissant le temps de conception. Attendre la réalisation de l'architecture matérielle pour développer le logiciel et vérifier l'intégration logiciel/matériel (Figure 3.a) entraîne un temps de conception très long. Le prototype est une solution pour diminuer le temps de conception. En effet, un prototype peut être employé pour effectuer la vérification logiciel/matériel et développer le logiciel avant que le circuit ne soit disponible (Figure 3.b), mais le développement d'un prototype spécifique à l'application requiert lui aussi beaucoup de temps et d'effort. On peut réduire ce temps et cet effort en utilisant une plateforme reconfigurable, ce qui revient à modifier légèrement l'architecture de la plateforme pour faire fonctionner chaque nouvelle application (Figure 3.c).

1.5 Contributions

1.5.1 Définition d'un flot de prototypage basé sur une plateforme reconfigurable

Pour réduire le temps et l'effort de développement du prototype, nous proposons une solution basée sur l'utilisation d'une plateforme reconfigurable et un flot pour transposer l'architecture du système monopuce sur cette plateforme reconfigurable. L'approche proposée permet de prototyper systématiquement des applications. Le processus prend en entrée un modèle de niveau RTL de l'application. L'application et la plateforme reconfigurable doivent être adaptées pour obtenir le prototype. Nous décomposons le processus de prototypage en quatre étapes: (1) assigner chaque partie de l'application à la plateforme, (2) configurer la plateforme pour être aussi proche que possible de l'architecture du système, (3) adapter l'architecture du système (si c'est encore nécessaire), et (4) cibler (compilation, synthèse, routage, …) l'architecture adaptée sur la plateforme configurée. Les solutions proposées tiennent compte des contraintes typiques des plateformes reconfigurables existantes.

1.5.2 Automatisation du flot du prototypage

La recherche d'une automatisation du flot de prototypage est une contribution importante, puisque le but est d'une part de facilité la tâche du concepteur en lui évitant des tâches difficiles et répétitives, et d'autre part de réduire le temps d'obtention du prototype. La configuration de la plateforme et l'adaptation de la description du système sont les deux étapes auxquelles nous nous sommes intéressés.

L'adaptation est une étape très difficile, car elle demande de la part du concepteur de modifier le code de bas niveau de l'application. Mais cette étape est facilitée par l'utilisation d'outils de génération automatique développés au sein de l'équipe SLS pour la conception des SoC. On peut donc s'appuyer sur ces outils de production pour adapter le code à la plateforme, ce qui revient à modifier la spécification de ce code en tenant compte des caractéristiques de la plateforme.

1.5.3 Proposition d'une solution de co-émulation basée sur une plateforme reconfigurable

Pour résoudre le problème de non disponibilité de composants physiques, nous pouvons combiner la simulation, l'émulation, et l'utilisation de vrais composants (co-

émulation). Une co-émulation employant la plateforme reconfigurable est une solution bon marché pour la vérification matériel/logiciel. Cette technique est appliquée en utilisant la plateforme ARM Integrator et un simulateur SystemC.

1.6 Plan du mémoire

La suite de ce mémoire est composée de six chapitres :

- Le chapitre 2 présente les techniques de vérification qui sont utilisées dans la conception de systèmes multiprocesseurs monopuces, ainsi que leurs intérêts, leurs points forts, et leurs défauts.
- Le chapitre 3 définit le flot de prototypage sur une plateforme reconfigurable. Nous discutons du modèle utilisé, du cas idéal, et du flot typique.
- Le chapitre 4 décrit le prototypage d'applications (VDSL et DivX).
- Le chapitre 5 porte sur l'évaluation des applications du chapitre 4. Ce chapitre donne une formalisation de l'automatisation du flot de prototypage et des perspectives autour de la co-émulation.
- Enfin, chapitre 6 conclut ces travaux et donne des perspectives.

CHAPITRE 2 : VERIFICATION DES SYSTEMES MONOPUCES : TECHNIQUES ET COMPARAISON

2.1 Introduction **12**

2.2 Objectif de la vérification d'un système monopuce **12**
2.2.1 Vérifier la spécification fonctionnelle 13
2.2.2 Vérifier l'architecture du système 13
2.2.3 Vérifier l'implémentation des composants du système 13
2.2.4 Vérifier l'intégration de composants hétérogènes 14
2.2.5 Détecter tous les bogues avant la fabrication 14

2.3 Difficultés de la vérification d'un système monopuce **15**
2.3.1 Hétérogénéité 15
2.3.2 Complexité 15
2.3.3 Il n'existe pas encore d'outils de déboguage de systèmes multiprocesseurs 16

2.4 Diverses techniques de vérification des systèmes monopuces **16**
2.4.1 Critères de comparaison des techniques de vérification des systèmes monopuces 16
2.4.2 La vérification formelle 18
2.4.2.1 Coût 19
2.4.2.2 Modèle 19
2.4.2.3 Flexibilité 19
2.4.2.4 Utilité 20
2.4.3 La simulation sur ordinateur 20
2.4.3.1 Coût 22
2.4.3.2 Modèle 22
2.4.3.3 Flexibilité 23
2.4.3.4 Utilité 23
2.4.4 L'émulation 23
2.4.4.1 Coût 24
2.4.4.2 Modèle 25
2.4.4.3 Flexibilité 25
2.4.4.4 Utilité 25
2.4.5 Le prototypage 25
2.4.5.1 Coût 27
2.4.5.2 Modèle 27
2.4.5.3 Flexibilité 27
2.4.5.4 Utilité 28
2.4.6 Synthèse des différentes techniques 28

2.5 Le prototypage basé sur une plateforme reconfigurable **30**
2.5.1 Intérêt du prototypage pour les systèmes monopuces 30
2.5.2 Intérêt du prototypage basé sur une plateforme reconfigurable 30
2.5.3 Intérêt de la co-émulation basée sur une plateforme reconfigurable 31
2.5.4 Exemples de flot de prototypage industrielle 31
2.5.4.1 Aptix 31
2.5.4.2 Flexbench 35

2.6 Conclusion **37**

2.1 Introduction

Ce chapitre introduit dans un premier temps les objectifs de la vérification d'un système monopuce tout au long du flot de conception. Les difficultés et les différentes techniques de vérification sont ensuite développées, en insistant plus particulièrement sur le prototypage sur une plateforme reconfigurable. La dernière partie de ce chapitre décrit quelque méthodes et flot de prototypage issus de technologies industrielles.

2.2 Objectif de la vérification d'un système monopuce

Dans la création des systèmes numériques, on doit s'assurer que les produits finaux fonctionnent correctement. La création de ces produits consiste en deux phases [WEB99C] principales : la conception et la fabrication (Figure 4). Ces deux phases exigent chacune un processus de vérification : la vérification de la conception (en anglaise : *design verification*) et le test ou diagnostic de la fabrication (en anglaise : *manufacturing test and diagnosis*). Le premier processus est là pour s'assurer que le système est correctement conçu, c'est-à-dire que le système correspond exactement à ce que l'on veut. Le deuxième permet de s'assurer que le produit fabriqué est identique au système conçu.

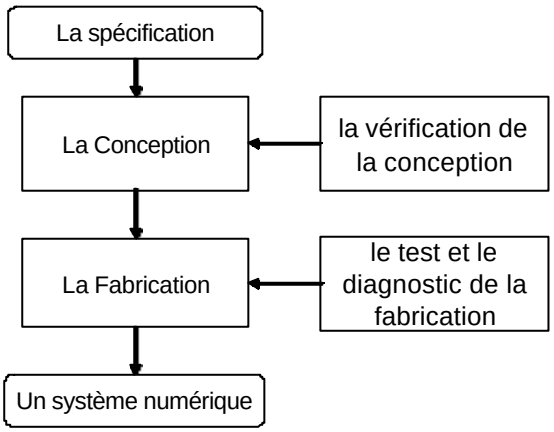


Figure 4 : Création d'un produit numérique

L'objectif du groupe SLS étant de concevoir des systèmes monopuces, on s'intéresse uniquement à la partie vérification de la conception. La vérification s'applique à chaque étape de la conception, et plus particulièrement pour vérifier la spécification fonctionnelle, l'architecture du système, l'implémentation des composants, l'intégration des composants et le système complet avant sa fabrication. Ces cinq critères sont abordés dans les paragraphes suivants.

2.2.1 Vérifier la spécification fonctionnelle

A partir de la spécification du système, on obtient une spécification fonctionnelle en développant une solution. Elle consiste à développer les algorithmes et à définir les représentations de données, ce qui correspond à une première optimisation. Les solutions retenues vont affecter tout le processus de conception et la performance du système. Ainsi, l'optimisation a un effet beaucoup plus significatif à cette étape que si elle est effectuée à bas niveau tel qu'au niveau porte logique.

Il faut vérifier aussi que toutes les contraintes fonctionnelles sont respectées. Il y a deux types de contraintes fonctionnelles : celles qui sont imposées par l'environnement et celles exigées par les clients/utilisateurs. Un exemple de contraintes imposées par l'environnement est le protocole utilisé pour connecter le système à un réseau existant. Un exemple de contraintes exigées par l'utilisateur est la compatibilité avec une ancienne version. Mais il faut tenir compte de ces contraintes dès le début de la conception.

2.2.2 Vérifier l'architecture du système

Quand une bonne spécification fonctionnelle est trouvée, les concepteurs cherchent une architecture pour l'implémenter. Si plusieurs architectures sont possibles, les concepteurs ont besoin d'évaluer la qualité et de vérifier la performance de chaque architecture. Cette étape correspond à l'exploration d'architecture.

A cette étape, le découpage de l'algorithme est fait. Les composants sont choisis pour exécuter chaque fonction. La spécification de chaque composant est définie. Il reste à vérifier que les autres contraintes (non fonctionnelles) sont respectées. Les contraintes sont par exemple : la fréquence maximum, la surface maximum autorisée, la consommation d'énergie, …

2.2.3 Vérifier l'implémentation des composants du système

Quand la spécification de chaque composant est fixée, les composants sont développés (l'implémentation). Les développements sont effectués par des équipes différentes, mais on peut aussi réutiliser des composants existants. La fonctionnalité de tous les composants doit être vérifiée individuellement, pour s'assurer qu'ils respectent bien les spécifications. Même s'ils seront vérifiés après l'intégration, la vérification de chaque composant est très

importante, car il est beaucoup plus facile de localiser la source d'un bogue à ce niveau qu'après l'intégration.

Du fait de la provenance variée des composants, il faut aussi vérifier la compatibilité entre ces composants (protocoles et échanges de données entre les composants par exemple). Dans le cas contraire, la performance du système peut fortement baisser.

2.2.4 Vérifier l'intégration de composants hétérogènes

L'étape suivante dans la conception est l'intégration de ces composants. C'est une étape très délicate, du fait que les composants peuvent provenir de sources différentes. Il faut développer des interfaces de communication pour permettre l'échange de données et la synchronisation entre les composants. Ces interfaces comprennent des parties matérielles et des parties logicielles.

A cette étape, les concepteurs vérifient : (1) les interfaces développées, (2) la fonctionnalité des composants et la compatibilité entre les composants, et (3) la performance du système après l'intégration. Les fonctionnalités et les compatibilités des composants sont vérifiées encore une fois car c'est souvent pendant l'intégration que les bogues apparaissent. La performance du système après l'intégration est mesurée car les interfaces utilisées entraînent des délais qui peuvent réduire les performances.

2.2.5 Détecter tous les bogues avant la fabrication

Quand toutes les étapes de la conception sont effectuées, une dernière vérification est nécessaire avant la fabrication. Il s'agit de mettre en œuvre le système complet pour vérifier que tous les aspects de ce système sont conformes à la spécification dans toutes les conditions. Normalement, une longue séquence de test est appliquée.

Cette étape de vérification est la dernière étape avant la fabrication. Ce qui signifie que les erreurs non détectées seront présentes sur le système réel. Les erreurs non détectées avant la fabrication augmente considérablement le coût du système et retarde la mise sur le marché d'une version non boguée.

Dans cette dernière étape de vérification, les concepteurs testent le système dans son environnement physique. Cet environnement est celui ou il sera utilisé (par exemple, on teste un prototype de téléphone portable sur un réseau GSM). Si c'est possible, le système doit être testé par un futur utilisateur. L'environnement physique et le comportement de

l'utilisateur en manipulant le système complexe sont souvent imprévisibles, ce qui est difficile à réaliser avec une séquence de test.

2.3 Difficultés de la vérification d'un système monopuce

2.3.1 Hétérogénéité

Une des difficultés dans la vérification d'un système monopuce provient de l'hétérogénéité des objets ou concepts manipulés. En effet, il faut vérifier les composants physiques mais aussi les programmes (système d'exploitation, application, interface utilisateur) qui s'exécutent sur les composants programmables. Et tous ces objets possèdent leurs propres critères à vérifier.

Mais certains compromis sont difficiles à évaluer. Pour illustrer ceci, supposons un système qui traite les commandes reçues d'un utilisateur. Si la spécification indique que le système doit répondre rapidement, les concepteurs diminueront la latence des interruptions. Mais si ce même système, dans sa partie traitement du signal à haut débit ne doit pas être interrompu dans une partie critique du code, alors une interruption attendra l'exécution d'une partie du code moins critique pour être prise en compte. Comment vérifier dans ce cas le compromis fait par les concepteurs ?

Les difficultés peuvent venir aussi de la nature des composants, surtout, si ils ont été développés par plusieurs équipes. Par exemple, un nom différent donné à un même signal devient une source de confusion pour la vérification.

Les niveaux d'abstraction entraînent aussi des difficultés. Si les composants d'un système sont décrits à différents niveaux d'abstraction (description comportementale d'un IP pour la simulation, modèle synthétisable pour d'autres), la simulation devient une simulation mixte (ou co-simulation) qui n'est pas toujours facile à mettre en œuvre.

2.3.2 Complexité

La complexité des systèmes monopuces est aussi une source de difficulté. Les erreurs dans une grande quantité de matériel sont très difficiles à trouver car les concepteurs ne peuvent observer qu'un nombre limité de signaux en même temps. Même si les concepteurs connaissent les signaux à observer, ils ne peuvent pas observer ces signaux pendant une très longue période car les outils n'enregistrent l'évolution de ces signaux que pendant un temps

limité. Le déboguage du matériel est un jeu qui consiste à observer les bons signaux au bon moment.

Quand une erreur est détectée sur un signal, il faut trouver l'origine du problème. Pour se faire, on sélectionne un nouvel ensemble de signaux et on recommence la séquence de vérification avec les mêmes vecteurs de test.

Les logiciels complexes sont aussi difficiles à déboguer et coûtent un temps important. Le parallélisme de tâches, les exceptions, et la dépendance d'exécution du code à des données rendent la reproduction d'une erreur très difficile. On a alors besoin de nombreux vecteurs de test pour reproduire cette erreur, et il faut une exécution pas à pas pour trouver la source d'erreur.

A cela s'ajoute le fait qu'un système complexe offre beaucoup de possibilités d'utilisation. Comme le comportement de l'utilisateur peut être imprévisible, il est impossible de créer des séquences de test correspondant à toutes les utilisations possibles.

2.3.3 Il n'existe pas encore d'outils de déboguage de systèmes multiprocesseurs

Même si dans le flot de conception, les parties matérielles et logicielles peuvent être développées en même temps, en réalité, il est difficile de développer le logiciel sans avoir le matériel pour l'exécuter. De plus, on a besoin d'outils pour contrôler et observer l'exécution des programmes. Comme ces outils n'existent pas encore, les concepteurs doivent utiliser des outils de déboguage de systèmes monoprocesseurs. Et cette technique n'est pas très efficace.

2.4 Diverses techniques de vérification des systèmes monopuces

2.4.1 Critères de comparaison des techniques de vérification des systèmes monopuces

La plus grande différence entre un système monopuce et un ASIC traditionnel est la grande quantité de logiciel exigée par le système. Ce logiciel inclut le SE, les pilotes de périphériques, la partie logicielle de l'interface de communication, et le logiciel de l'application. Pour vérifier le logiciel lors du développement, il existe plusieurs techniques. Par exemple, on peut utiliser la simulation pour valider le développement de certains pilotes de périphériques de bas niveau avec des performances de l'ordre de quelques centaines de kilohertz. Cependant, pour les autres parties de logiciel, il est nécessaire de mettre en œuvre

d'autres techniques avec d'autres caractéristiques pour vérifier le système avant fabrication et pour vérifier l'intégration du logiciel sur le matériel.

Dans ce chapitre, nous décrivons quatre techniques de vérification qui utilisent différents modèles. Ces techniques sont la vérification formelle, la simulation, l'émulation, et le prototypage. La discussion est abordée du point de vue théorique. Dans l'utilisation pratique, ces techniques sont combinées et les termes sont très souvent mélangés. Par exemple, ce qui s'appelle la simulation avec un accélérateur de matériel est dans la pratique une combinaison de technique de simulation et d'émulation.

Il y a quatre caractéristiques principales permettant de comparer des techniques de vérification de systèmes monopuces : coût, modèle, flexibilité, et utilité.

Le coût en terme d'argent, de temps, et d'effort est naturellement la caractéristique la plus importante puisque toutes ces techniques visent à réduire le coût. Il y a principalement deux facteurs qui constituent le coût: (1) le coût pour établir le modèle, et (2) le coût du procédé de vérification qui utilise ce modèle.

D'une façon générale, le coût pour détecter un problème plus tôt dans la conception et pour le corriger est beaucoup moins élevé que si il est découvert plus tard. Le coût pour résoudre un problème matériel est beaucoup plus élevé que pour un problème logiciel. Souvent, on préfère résoudre un problème matériel en utilisant une solution logicielle quand c'est possible.

Le modèle est la caractéristique principale qui différencie ces différentes techniques. La vérification formelle emploie des notations formelles. La technique de simulation emploie des modèles d'exécutions et des modèles de calcul sur ordinateur. L'émulation emploie des modèles physiques qui imitent le comportement du matériel. Et le prototypage combine ces divers modèles et composants matériels.

La flexibilité correspond à la facilité de changer, d'adapter ou de modifier le modèle du système. Elle est très importante, particulièrement quand la conception n'est pas encore trop avancée, et que des modifications sont nécessaires. Normalement, la technique est plus flexible quand il faut moins d'effort et de ressources pour arriver au modèle. Il est facile de comprendre que le logiciel est beaucoup plus flexible que le matériel. Par conséquent, une technique utilisant des modèles logiciels est beaucoup plus flexible.

Par exemple, supposons que des développeurs de logiciels détectent un problème qui est très difficile à résoudre avec un nombre de registres donné, mais qui devient trivial si au moins un registre supplémentaire est disponible. Si la puce est fabriquée, les développeurs n'ont pas d'autres alternatives que de corriger le problème avec une solution logicielle, même si celle ci dégrade les performances. Au contraire, si le logiciel est développé sur une architecture matérielle flexible, le registre supplémentaire est ajouté et le problème est résolu plus facilement.

L'utilité d'une technique est déterminée par les facteurs suivants : précision, vitesse, observabilité, et objectif de la vérification. Les trois premiers facteurs (précision, vitesse, et observabilité) déterminent si une technique convient à un objectif donné. Pour un objectif donné, il y a un niveau approprié de ces trois facteurs requis.

La précision est associée à l'exactitude, à la granularité des événements, et à la granularité du temps. L'observabilité est associée à la capacité d'observer et de contrôler l'état du modèle. La vitesse indique le temps nécessaire pour effectuer la vérification du modèle.

La suite de cette section décrit les quatre techniques que sont la vérification formelle, la simulation, l'émulation et le prototypage du point de vue des caractéristiques précédemment citées.

2.4.2 La vérification formelle

Il y a deux types de vérification formelle [STA94] : le déboguage de la spécification et la vérification de l'implémentation. Le premier type vérifie si la spécification est bien décrite et si tous les besoins sont bien inclus. Le deuxième type vérifie si la spécification est bien implémentée.

Jusqu'à présent, la technique de vérification formelle n'est pas encore largement utilisée dans la conception de systèmes monopuces. Les obstacles principaux sont [ROS98] :

(1) La complexité du processus de vérification est très grande donc elle est utilisée dans des cas simples.

(2) Cette technique a besoin d'interactions entre les concepteurs et les outils, donc l'automatisation est le point faible de cette technique. Ce point faible implique que cette technique est difficile à utiliser pour les systèmes complexes.

(3) Il est difficile de prendre en compte l'environnement.

(4) Les techniques utilisées pour le matériel et le logiciel sont différentes à cause de leurs caractéristiques différentes. Le matériel est traité en tant que système parallèle synchrone. Par exemple, en vérification formelle, le matériel est décrit en utilisant les «*binary decision diagram (BDD)*». Les procédures de vérifications exploitent des régularités et des «localisations». Le logiciel est traité en tant que système asynchrone sériel. La technique de vérifications utilise une méthode «*partial order reduction*». La vérification matériel/logiciel est très rarement effectuée car le nombre d'étapes nécessaire pour vérifier le matériel et pour vérifier le logiciel est très différent. On peut avoir besoin de cent étapes de vérification du matériel pour chaque étape du logiciel.

2.4.2.1 Coût

Le coût financier est faible, mais il est élevé en terme d'effort et de temps, particulièrement si le système est complexe. Ce problème existe car il n'y a pas d'automatisation pour prouver l'exactitude du système.

2.4.2.2 Modèle

La vérification formelle utilise un ensemble de notations prédéfinis pour construire le modèle. Ce modèle s'appelle le modèle formel. L'utilisation d'une notation formelle donne une précision élevée de la description du système. Malheureusement, elle est très complexe donc difficile en mettre en œuvre.

La vérification formelle utilise des modèles différents pour le matériel et le logiciel. Elle utilise les BDD pour le matériel et des modèles asynchrones sériels pour le logiciel (algorithme ou organigramme). Il existe quelques travaux [POLIS] pour traiter la vérification matériel/logiciel mais ils ne sont pas encore largement acceptés ou utilisés. Un autre problème de la vérification formelle est la difficulté à l'intégrer avec d'autres techniques de vérification car le procédé de vérification est différent. Au lieu de valider le système en utilisant des vecteurs de test, son fonctionnement correct doit être prouvé formellement.

2.4.2.3 Flexibilité

Le modèle formel est très flexible. Il peut être utilisé pour un large éventail de systèmes en utilisant une granularité variée. Le modèle est facile à modifier parce qu'il est

construit à la main. Il n'est pas nécessaire de passer par des procédés tels que la compilation ou la synthèse, donc les modifications sont faciles à faire.

2.4.2.4 Utilité

La vérification formelle est très rapide quand elle est appliquée à un petit système parce qu'avec cette technique, le système n'est pas validé en utilisant des vecteurs de test mais prouvé formellement. Mais, la preuve d'un système complexe est un travail long et fastidieux :

1. Modéliser un système complexe est long et difficile.
2. Il y a très peu d'automatisation dans le procédé de vérification donc des interactions massives entre l'outil et le concepteur sont nécessaires. Ceci implique qu'il est trop laborieux pour être appliqué aux systèmes complexes.

La précision de cette technique est très grande quand le système est décrit à un niveau d'abstraction bas. L'observabilité de la vérification formelle est plutôt bonne. Les états du système évalué peuvent être observés dans des situations variées. Pour un système simple, les trois caractéristiques sont élevées mais pour un système complexe, sa vitesse de vérification est très faible.

2.4.3 La simulation sur ordinateur

Littéralement, la simulation est une approche basée sur un modèle pour l'analyse des systèmes. Dans la conception des systèmes numériques, la simulation équivaut à tester le comportement des systèmes en les exposant à des vecteurs de test, sauf que les systèmes sont remplacés par un modèle sur ordinateur. Les modèles utilisés peuvent être des modèles de calculs ou des modèles d'exécution.

La simulation est la technique la plus largement employée pour vérifier les systèmes monopuces, les ASIC, et plus généralement les systèmes numériques et analogiques. Il y a divers types de simulation, ils s'étendent de la simulation comportementale de haut niveau à la simulation électrique/physique de très bas niveau. La précision et la vitesse de ces simulations sont également variées. La simulation de plus bas niveau donne une excellente précision jusqu'au temps physique de l'ordre de la pico seconde, mais à une vitesse très réduite. La simulation de niveau plus élevé donne une précision moins bonne mais avec un temps d'exécution plus rapide. Donc, il y a toujours un compromis entre vitesse et précision.

Il y a beaucoup de travaux qui combinent ces différents types de simulation. Généralement, les concepteurs combinent ces différents types de simulation pour profiter des avantages de chaque niveau de simulation.

Il existe plusieurs classifications de niveau de simulation. Le nombre de niveaux et la frontière entre ces niveaux sont différents, mais à un niveau élevé correspond une abstraction élevée, donc une granularité moins fine et bien sur une précision moins bonne, mais la vitesse d'exécution est plus rapide.

Par exemple, nous présentons six niveaux d'abstractions de la simulation [CLO02] pour illustrer le compromis dans la simulation :

Le niveau fonctionnel modélise le comportement prévu du système, sans précision sur la façon dont il sera réalisé. Ce niveau de simulation aide *à comprendre* le système. Un exemple de ce niveau de simulation est l'exécution de la spécification de l'application. Cette simulation aide les concepteurs à analyser les besoins du système.

Le niveau architectural saisit toute l'information nécessaire pour programmer le matériel. Ce niveau de simulation aide les concepteurs à vérifier le logiciel pour le système, particulièrement pour les parties qui sont indépendantes du matériel. L'exemple de cette simulation est la simulation du modèle au niveau transactionnel (TLM). Ce niveau de simulation peut être employé pour commencer le développement du logiciel (pour la partie indépendante de matériel) et pour l'exploration d'architecture.

Le niveau micro architecture permet de *faire une simulation avec une précision au niveau cycle*. Ce niveau de simulation est employé par les concepteurs pour commencer le développement des pilotes de bas niveau, et pour mesurer les performances du système.

Le niveau RTL est évidemment un niveau d'abstraction qui décrit le système comme un ensemble de registres et de relations logiques entre les registres avec des notions de temps précis au cycle prés. Ce niveau est difficile à distinguer du niveau micro architecture puisque normalement pour simuler au niveau cycle, le système doit être décrit au niveau RTL. L'objectif principal de ce niveau est le point d'entrée des outils pour *implémenter* la partie matérielle.

Le niveau porte logique décrit le système sous forme de portes logiques et de registres. Ce niveau est utilisé par les outils de synthèse pour *optimiser le matériel*.

Normalement, les concepteurs de matériel ne travaillent pas à ce niveau. Ce sont les outils qui raffinent l'application de niveau RTL à ce niveau.

Le niveau de *layout* (dessin au micron) est le niveau le plus bas de simulation. Ce niveau d'abstraction est employé pour extraire les paramètres physiques et les paramètres électriques du matériel. La technologie d'implémentation est prise en compte à ce niveau d'abstraction

Comme conclusion, il faut noter que : (1) le concepteur ne peut pas avoir une vitesse et une précision élevées en même temps et également, (2) qu'il est difficile d'intégrer l'environnement physique.

2.4.3.1 Coût

Les coûts financiers de simulation sont très divers. Ces coûts correspondent au coût de l'ordinateur, le coût des outils de simulation, le coût de construction du modèle. Si le coût des machines est indépendant du type de simulation, le coût des outils contribue fortement au coût financier. Ce coût est varié et dépend au niveau d'abstraction et des outils ou vendeurs choisis.

L'effort et le temps nécessaire pour effectuer la simulation dépendent du niveau de la simulation. La simulation de plus bas niveau nécessite un effort plus élevé que les niveaux plus hauts, parce que le modèle utilisé est plus difficile à construire (on a besoin d'écrire un modèle de bas niveau ou d'extraire des paramètres pour ce modèle). Le temps nécessaire pour effectuer une simulation de bas niveau est plus long car le modèle utilisé est plus complexe.

2.4.3.2 Modèle

Les modèles utilisés pour la simulation sont variés selon de niveaux d'abstraction utilisés. Les langages de programmation (ou leurs extensions) sont utilisés pour décrire les modèles de haut niveau d'abstraction (niveau fonctionnelle), niveau architecture, et niveau micro architecture). Au niveau fonctionnel, on décrit le comportement du système dans le modèle. Au niveau architecture et au niveau micro architecture, on décrit le modèle du système comme un ensemble des composants interconnectés.

Au niveau RTL et au niveau porte logique, les systèmes sont décrits dans un langage de description de matériel (par exemple : VHDL, Verilog, EDIF).

Au plus bas niveau (*layout*), on décrit le système comme un ensemble de polygones en plusieurs couches. Afin de simuler ce système, on doit extraire le circuit et ses paramètres. Le résultat de cette extraction est un fichier qui décrit le système comme un circuit électrique composé de composants électroniques élémentaires (transistor, diode, condensateur, inducteur, résistance, etc.).

2.4.3.3 Flexibilité

La simulation est une technique très flexible car il est facile de modifier un modèle de simulation. De plus, on peut combiner des modèles de différents niveaux.

2.4.3.4 Utilité

Selon les niveaux d'abstraction, vitesse et précision sont différentes. Le point faible de la simulation est qu'une vitesse élevée et une précision élevée ne peuvent pas être obtenues en même temps. L'observabilité de la simulation est bonne car on peut récupérer toute l'évolution des états des signaux du système simulé. Pour la simulation de bas niveau, le point faible est la vitesse tandis que pour la simulation de haut niveau, le point faible est sa précision.

2.4.4 L'émulation

Le terme « émulation » est normalement associé à l'émulation logique, mais il est aussi utilisé dans un sens différent dans le domaine de l'électronique numérique. Par exemple, un émulateur en circuit (« In-Circuit Emulator » / ICE) est un outil de déboguage qui remplace un microprocesseur par une prise et un câble reliés au PC.

La technique d'émulation est employée dans deux cas :

(1) Pour vérifier de comportement du matériel émulé en observant les entrées/sorties, typiquement en utilisant une technologie basée sur des FPGA.

(2) Pour vérifier d'autres parties du système. On peut observer les comportements des autres parties du système en regardant ses interactions avec le composant émulé. Par exemple avec un «processor-ICE », les concepteurs vérifient le logiciel exécuté par le processeur émulé.

L'émulation présente deux avantages par rapport à la simulation : elle permet un nombre de vecteurs de test plus grand grâce à sa vitesse d'exécution plus rapide, et elle permet une connexion à l'environnement physique. Mais il existe aussi quelques points

faibles de l'émulation. Il est difficile d'observer l'état des signaux internes, une erreur temporelle est difficile à détecter, l'effort demandé est plus important (flot de compilation/synthèse), le coût plus élevé (un émulateur coûte relativement cher) et il faut décrire le système à un bas niveau. A cause de ces caractéristiques, l'émulation est employée après plusieurs étapes de simulation (quand la conception du matériel est déjà avancée).

L'environnement d'émulation se compose de trois parties [WEB99b] : le matériel programmable (l'émulateur), un compilateur, et le logiciel pour observer et contrôler l'émulateur. Le matériel programmable agit comme un modèle du système à émuler. Le compilateur accepte une description du système à émuler et il configure l'émulateur pour modéliser le système. Le logiciel d'instrumentation et de contrôle est employé pour commander et observer les entrées/sorties de l'émulateur.

Il existe plusieurs types de matériels utilisés [ROS98] pour l'émulation :

(1) l'émulateur basé sur un processeur. Ce type d'émulateur est composé d'un processeur et d'un ensemble de registres. Les registres gardent l'état en cours du système. Le processeur sert à calculer le prochain état du système et les sorties du système. *Quickturn CoBalt* [SAW96] et *Synopsys Arkos* sont des émulateurs basés sur un processeur.

(2) l'émulateur basé sur des FPGA. [ROS98] contient une illustration de plusieurs types d'émulateurs basées sur des processeurs et des FPGA. Un exemple d'émulateur basé sur FPAG est *Mercury system* de Quickturn.

(3) l'émulateur basé sur des composants programmables spécifiquement conçus pour l'émulation. Des exemples de ce type d'émulateur sont *Aptix prototyping system* [APT02] et *Celaro* de *Mentor Graphics* [MEN00]. Aptix utilise ses composants spécifiques appelés FPIC (en Anglais : Field Programmable Interconnect Component) pour construire un émulateur avec plusieurs FPGA. Mentor Graphics développe aussi des composants spécifiques pour son émulateur (Celaro et VStation).

2.4.4.1 Coût

Le coût financier de l'émulation est très élevé. Le coût d'un émulateur logique performant est de l'ordre de plusieurs centaines de milliers d'euros. L'effort et le temps nécessaires pour émuler un système sont aussi élevés car les concepteurs doivent développer

un modèle du système qui respecte les contraints de synthèse/compilation, optimisation, placement et routage.

2.4.4.2 Modèle

L'émulation utilise un modèle physique pour imiter le matériel émulé. Les implémentations de ces modèles sont variées. Essentiellement, il y a deux types d'implémentation d'émulateur : en utilisant du matériel configurable, et en employant un processeur. Le point de départ de la plupart des techniques d'émulation est un modèle de niveau RTL ou portes logiques.

2.4.4.3 Flexibilité

L'émulation est moins flexible que la simulation car il faut un modèle du système à concevoir décrit à un bas niveau d'abstraction. Il faut aussi toujours passer par les étapes de compilation/synthèse et placement et routage.

2.4.4.4 Utilité

L'observabilité de l'émulation est très faible puisque on ne peut observer que les entrées-sorties du système. L'émulation est beaucoup plus rapide que la simulation. La précision de l'émulation est très bonne (précis au cycle prés) car le système est décrit au niveau RTL. La vitesse et la précision sont élevées mais l'observabilité reste faible.

2.4.5 Le prototypage

Dans la conception de systèmes monopuces, le prototypage est une réalisation préliminaire d'un système sur une cible différente de la cible de réalisation finale [HAR97a][CHA99]. Tous les composants de l'architecture doivent être inclus dans le prototype. La réalisation de ces composants peut être diverse : vrais composants logiciel/matériel, composants émulés, ou même composants simulés. Il existe aussi des travaux pour intégrer la vérification formelle sur prototypage [KOR01]. Basée sur la réalisation de ces composants, on peut définir plusieurs types de prototypage:

a. Le prototypage virtuel

L'implémentation d'un prototype virtuel se fait par simulation de tous les composants qui constituent le système. On trouve également le terme de co-simulation, car fondamentalement, il s'agit d'une simulation avec plusieurs simulateurs. On combine

alors dans cette technique les simulations basées sur des ISS, des simulations de niveau RTL, des simulations de niveau fonctionnel, et des simulations de niveau transactionnel.

Le point faible d'un prototype virtuel est sa vitesse. En effet, on utilise les simulations pour implémenter les composants, donc la vitesse n'est pas aussi élevée qu'en utilisant des émulateurs ou des composants physiques. En plus, il est difficile de connecter un prototype virtuel avec l'environnement physique du système.

L'avantage est sa flexibilité et son coût puisqu'il s'agit de simulation. Ce qui permet aussi d'introduire des modèles de haut niveau.

b. Le prototypage matériel

Le prototypage matériel utilise des modèles physiques de composants pour réaliser le système. Il existe deux types de prototype matériel : le prototype spécifique à l'application et le prototype basé sur une plateforme reconfigurable.

- **Le prototype spécifique à l'application**

  L'implémentation de ce prototype emploie des composants matériels spécifiques. Ce prototype est dédié spécifiquement à la modélisation d'une application spécifique. Chaque module matériel dans le système est remplacé par une composante physique. Les connexions sont réalisées par des fils ou des liaisons sur un circuit imprimé.

  Les points faibles sont son coût et sa flexibilité. Il faut plus de temps et plus d'effort pour développer ce type de prototype que les autres. De plus, si les composants requis ne sont pas toujours disponibles, ce prototype ne peut pas être développé. Il n'est pas flexible, et les modifications sont difficiles à effectuer si l'on modifie le système.

  L'avantage de ce prototype est sa performance puisque qu'on peut obtenir un prototype dont l'architecture est très proche de celle du système final.

- **Le prototype reconfigurable**

  L'implémentation de ce prototype est une plateforme reconfigurable. On utilise des composants configurables pour implémenter les composants physiques et les connexions.

  Le point faible de ce type de prototype est le point d'entrée, puisqu'il faut une description de niveau RTL ou de niveau porte logique de tous les composants.

Malheureusement, les descriptions de niveau RTL pour implémenter les parties matérielles ne sont pas toujours disponibles.

L'avantage de ce prototype est qu'il est beaucoup moins cher en terme d'effort, d'argent, et de temps nécessaire pour la mise en oeuvre grâce aux parties matérielles physiques qui sont déjà développées.

c. Le prototypage mixte

L'implémentation d'un tel prototype consiste à combiner la simulation, l'émulation, et l'utilisation de composants réels. Une des difficultés est de réaliser les connexions entre les différents ordinateurs et les parties émulées. En effet, il faut mettre en œuvre un mécanisme efficace pour faire communiquer les composants réels et les simulateurs.

Les points faibles de la co-émulation sont sa vitesse et la difficulté à l'utiliser. La vitesse est limitée par la vitesse de simulation et par les performances des connexions. Cette technique demande aussi une connaissance multiple de la part des concepteurs.

L'avantage de la co-émulation repose sur un compromis entre simulation et émulation. La partie simulée correspond aux composants qui requièrent une grande flexibilité, une observabilité élevée ou un modèle de haut niveau. La partie émulée permet d'avoir une bonne vitesse d'exécution pour les autres composants.

2.4.5.1 Coût

Le coût du prototypage en terme d'argent, d'effort, et de temps dépend du choix d'implémentation. En général, un prototype reconfigurable, un prototype virtuel ou la co-émulation coûtent beaucoup moins cher en conservant une précision, une vitesse, et une observabilité acceptables.

2.4.5.2 Modèle

Le prototypage combine divers modèles pour profiter des avantages de chaque technique. On utilise des modèles de calcul pour les parties simulées, des émulateurs et des composants physiques pour les autres parties.

2.4.5.3 Flexibilité

La flexibilité du prototypage dépend de la façon dont il est implémenté. Un prototype reconfigurable, un prototype virtuel, et la co-émulation sont plus flexibles qu'un prototype spécifique.

2.4.5.4 Utilité

Le prototypage offre la possibilité de faire des compromis entre observabilité, vitesse, et précision. L'observabilité est obtenue soit en ajoutant une interface de déboguage au composant matériel soit par la simulation. La vitesse est obtenue par l'émulation et l'utilisation de composants physiques, et la précision est obtenue soit avec une simulation de bas niveau, soit à partir d'émulation du composant.

2.4.6 Synthèse des différentes techniques

Parmi les quatre techniques que l'on a comparées précédemment, chacune a ses propres caractéristiques. Chaque technique peut alors être utilisée efficacement dans une étape différente du flot de conception des systèmes monopuces.

La vérification formelle est une technique flexible et elle a une bonne utilité (bonne précision, bonne observabilité, et vitesse de vérification élevée). Malheureusement, elle est trop complexe à utiliser à bas niveau d'abstraction. De plus, il est difficile de l'utiliser pour vérifier du logiciel et du matériel en même temps. Cette technique est efficace au début de la conception pour vérifier la description de haut niveau.

La simulation est une technique très flexible qui offre une bonne observabilité. Cette qualité a permis la généralisation de son utilisation. Ainsi, aujourd'hui, on ne peut plus développer un système monopuce sans simulation. Le défaut principal de cette technique est l'impossibilité de concilier vitesse de simulation et précision. Son autre défaut est la difficulté à intégrer l'environnement physique. Cette technique est alors aussi lente pour le développement d'un logiciel complexe devant être exécuté par plusieurs processeurs.

A l'opposé, l'émulation est une technique qui permet d'allier une vitesse élevée et une bonne précision. C'est une bonne technique pour les tests dans un environnement physique. Les défauts de cette technique sont sa faible flexibilité, sa mauvaise observabilité, et son coût. Cette technique n'est alors utilisable que lorsque la conception est déjà bien avancée et que la majorité des bogues est déjà éliminée.

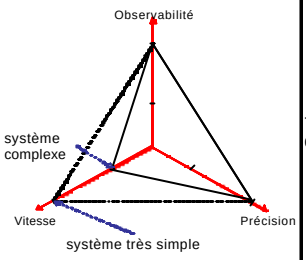
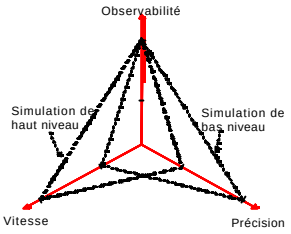
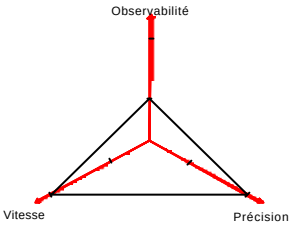
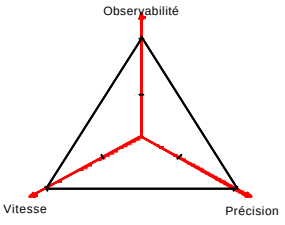
Technique	Coût	Modèle	Flexibilité	Utilité	Dans quel cas l'utiliser
Vérification formelle	- faible coût financier (ordinateur, outil de vérification formelle) - effort et le temps de vérification considérables	ensemble de notations prédéfinies	Très flexible		- verification la spécification des systèmes monopuces
Simulation	- faible coût financier (ordinateur, outil de simulation) - effort et temps faibles comparés aux autres techniques	modèle d'exécution/de calcul sur l'ordinateur	Très flexible		- toutes les étapes de conception des systèmes monopuces
Emulation	- élevé (financiers, temps, et efforts)	un modèle physique qui se comporte comme le composant émulé	Pas flexible		- vérification avec un trés long vecteur de test - vérification finale (quand le système n'a plus trop de bogues)
Prototype	- dépend de la réalisation du prototype	combine divers modèles pour profiter des avantages de chaque modèle	Pas flexible en matériel, et flexible avec la simulation		- quand la conception est déjà avancée (on n'a plus beaucoup de bogues)

Tableau 2 Tableau récapitulatif de technique de vérification

Le prototypage combine les avantages de chaque technique. On peut avoir une bonne observabilité, une bonne précision, ainsi qu'une vitesse élevée. De plus, on peut développer un logiciel sur le prototype et effectuer un test sur environnement physique. Le problème du prototypage se situe au niveau de son développement qui est difficile et coûteux. Le Tableau 2 résume des caractéristiques des ces techniques. L'utilité est représentée par un triangle unique en fonction des valeurs des trois facteurs que sont l'observabilité, la précision et la vitesse.

Afin d'accélérer la conception des systèmes monopuces et pour éviter l'apparition de bogues après fabrication, on a besoin de deux choses : un développement du logiciel plus tôt

et une vérification rapide permettant d'utilisation d'un long vecteur de test pour assurer une élimination complète de bogues. Le prototypage sur une plateforme reconfigurable et la co-émulation permettent de répondre à ces besoins avec un coût raisonnable.

2.5 Le prototypage basé sur une plateforme reconfigurable

2.5.1 Intérêt du prototypage pour les systèmes monopuces

Sur les cinq points décrits dans le paragraphe 2.1, le prototypage présente un intérêt pour quatre d'entre eux : on peut utiliser le prototypage pour évaluer l'architecture du système (point 2), pour vérifier la fonctionnalité et la compatibilité des composants (point 3), pour vérifier l'intégration des composants (point 4), et tester le système avant fabrication (point 5).

L'évaluation de l'architecture est faite en simulant chaque composant. A cette étape, on ne possède que l'architecture du système et les spécifications de chaque composant. On ne peut alors que simuler le comportement de ces composants. Aussi, seul un prototypage virtuel (co-simulation) est possible. On peut aussi faire une co-émulation si certains composants sont disponibles.

La vérification de la fonctionnalité et de la compatibilité des composants peut être faite par prototypage sous certaines conditions. Pour développer la partie logicielle, on peut utiliser le prototype matériel pour exécuter ces programmes. Pour développer un composant matériel, on peut utiliser une technique de co-émulation si d'autres composants sont déjà disponibles. On simule le composant développé et on le connecte avec d'autres composants.

On peut aussi vérifier l'intégration des composants, et tester le système complet en utilisant un prototype. Tous les composants sont prêts à cette étape. Donc, on peut développer un prototype spécifique ou un prototype basé sur une plateforme reconfigurable. Avec ce prototype, on teste le système dans son environnement physique. Pour cette vérification, les entrées utilisées sont les mêmes que celles que rencontrera le système final. Ce prototype permet de faire des tests d'une durée beaucoup plus longue.

2.5.2 Intérêt du prototypage basé sur une plateforme reconfigurable

Les avantages à utiliser une plateforme reconfigurable sont évidents. Le coût de prototypage est réduit. On ne doit pas financer le développent pour chaque application grâce

à la réutilisation du matériel. Le processus de prototypage est plus rapide car on n'a pas besoin de concevoir et de fabriquer le matériel physique. Ainsi, l'effort est réduit.

Grâce à ce développent plus rapide, les concepteur peuvent commencer le développement des parties logicielle plus tôt, ce qui raccourcit le temps de conception.

2.5.3 Intérêt de la co-émulation basée sur une plateforme reconfigurable

La co-émulation est utile quand les concepteurs ont besoin de vérifier le système complet et si ils n'ont pas encore tous les composants à leur disposition. Dans ce cas, les composants non existants peuvent être simulés, le reste des composants étant émulé.

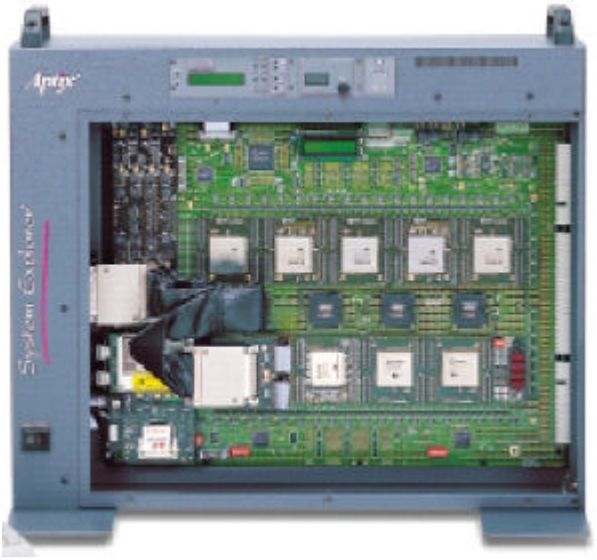
Pour observer précisément le comportement d'un composant, il est quelques fois utile d'utiliser la co-émulation. En effet, ce composant est alors simulé, ce qui permet d'étudier l'évolution des signaux internes, en émulant le reste du système pour des raisons de performance.

2.5.4 Exemples de flot de prototypage industrielle

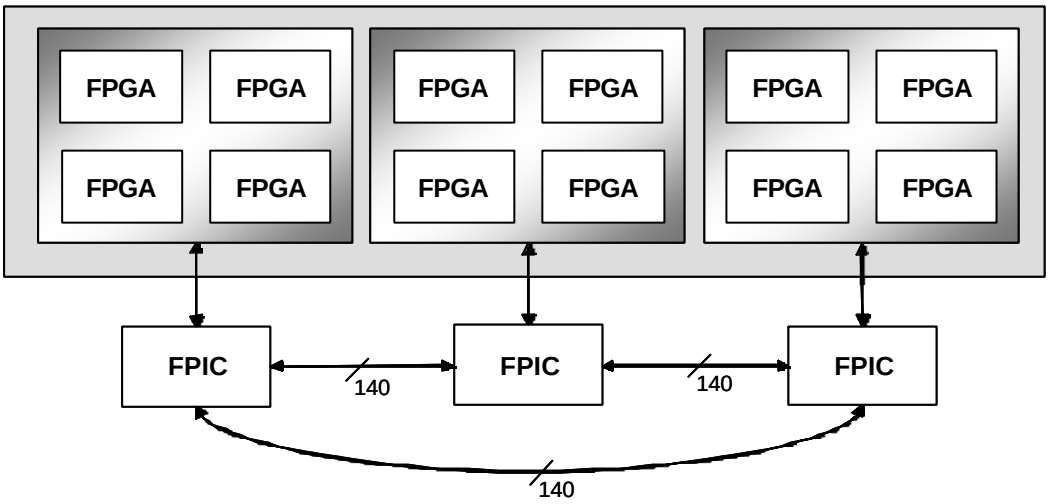
2.5.4.1 Aptix

Un exemple de flot de prototypage mis en œuvre dans le milieu industrielle est le flot de prototypage de Aptix Corp [APT02]. Aptix propose un flot de prototypage et un ensemble d'outil nommé *Aptix Prototype Studio*. Ces outils de prototypage visent à trois plateformes de prototypage : la plateforme *System Explorer*, la plateforme *Software Integration Station* et un *circuit imprime spécifique* (custom PCB). En plus, le flot permet aussi utilisation de co-émulation en utilisant la plateforme *System Explorer*.

System Explorer **(**Figure 5**)** est une plateforme de prototypage pour réaliser un prototype matériel. Cette plateforme est basée sur un composant d'interconnexion réconfigurable (FPIC). Ce composant permet d'obtenir des connexions configurables entre les cartes montées sur cette plateforme. Il permet aussi d'observer les signaux avec un analyseur logique. De plus, cette plateforme peut être connecté aussi à une simulation RTL pour effectuer une co-émulation. Aussi cette plateforme convient très bon pour le déboguage de matériel.



(a) Le matériel



(a) L'architecture des interconnexions

Figure 5 Le System Explorer d'Aptix

Software Integration Station est une plateforme basée sur FPIC aussi, mais sans la possibilité de déboguer du matériel. Elle n'a pas de connexions pour analyseur logique, ni d'interface de co-émulation. Le point fort de cette plateforme est sa vitesse très élevée. De plus, cette plateforme est beaucoup moins cher que le *System Explorer*. Cette plateforme est alors utilisée pour réaliser un environnement d'exécution (matériel) pour le développement du logiciel.

Aptix propose une gamme de plateforme de prototypage pour toutes les phases de la conception de système (Figure 6)

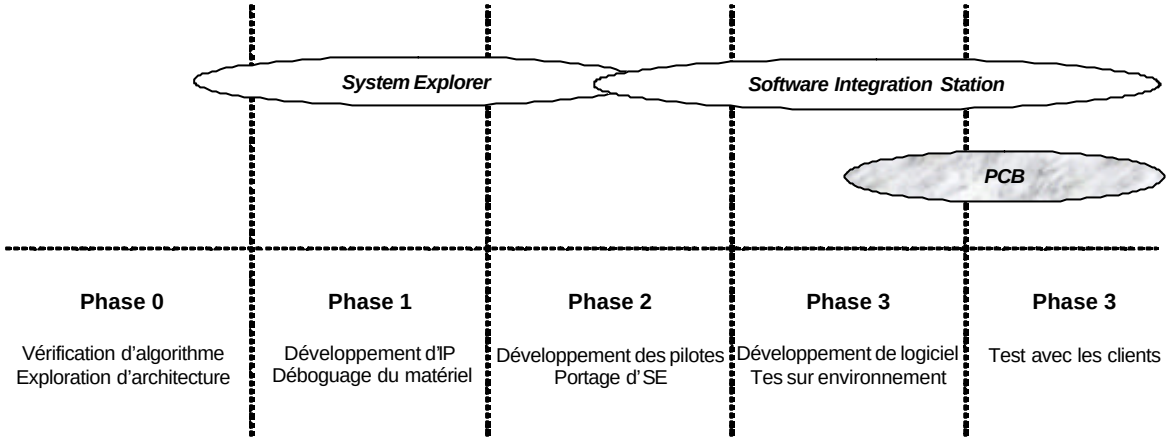


Figure 6 Les phases dans la conception des système monopuces d'Aptix

La phase 0 est la phase de conception de niveau système. Les concepteurs vérifient l'algorithme utilisé et explorent l'architecture. Dans cette phase, le coût de prototype n'est pas important. Les plus importantes caractéristiques dans cette phase sont la vitesse, l'observabilité, et la flexibilité car la description du système n'est généralement pas encore stable. Normalement, il y a beaucoup de modifications à ce niveau. La simulation ou le prototypage virtuel est alors préféré.

La phase 1 est la phase de conception classique du matériel. Comme la phase 0, le coût du prototypage est moins important que l'observabilité et la flexibilité. Mais, la vitesse dans cette phase n'est pas très importante car le vecteur de test dans cette phase n'est pas très long. L'Aptix propose alors d'utiliser *System Explorer*.

La phase 2 est la phase de développement du logiciel de bas niveau. La vitesse devient importante car le logiciel commence à être complexe. Parfois, on a encore besoin d'observer les caractéristiques matérielles. Aptix propose d'utilisation de *System Explorer* pour *co-émulation*, et puis de *Software Integration Station*.

La phase 3 est la phase de développement du logiciel de l'application. Les prototypes sont utilisés pour exécuter le logiciel. Dans cette phase, le coût devient important car plusieurs prototypes sont utilisés par les différentes équipes de conception. La vitesse devient importante aussi car les parties logicielles sont vraiment complexes. Par contre, la flexibilité et l'observabilité du matériel deviennent moins nécessaires car le matériel est déjà stable dans cette phase. L'utilisation de *Software Integration Station* est alors proposée.

Dans la phase 4, les prototypes sont testés par les utilisateurs. Des nombreux prototypes sont nécessaires. Le coût des prototypes est alors très important. La vitesse d'exécution doit être proche à la vitesse finale du produit. Par contre, on n'a pas besoin d'observabilité et de flexibilité. Les prototypages en utilisant *Software Integration Station* et les prototypages sur circuits imprimés convient bien.

Les outils de prototypage d'Aptix consistent en des plateformes matériels et des outils logiciels. Les plateformes matériels sont environnement d'exécution. Les outils logiciels sont pour automatiser le flot de prototypage : *Design Pilot*, *Explorer*, *FPGA Connector*, et *Module Verification Platform*, auxquels s'ajoutent les outils de synthèse pour FPGA et pour la conception des circuits imprimés (Figure 7).

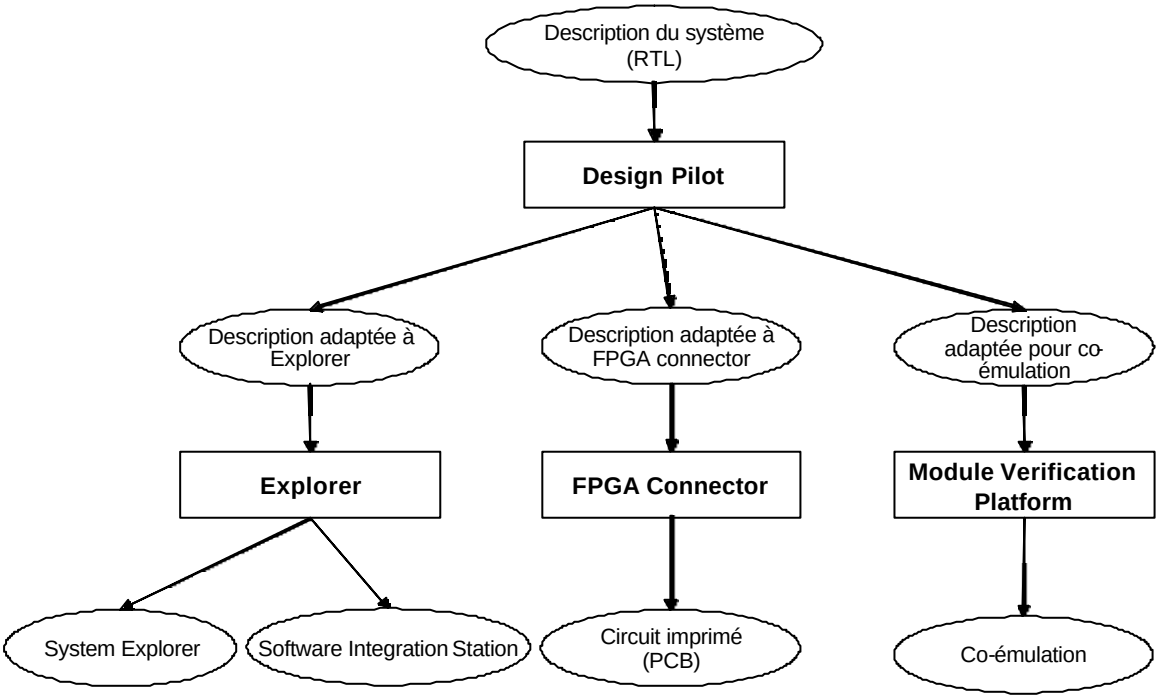


Figure 7 Flot de prototypage d'APTIX

Design Pilot est un logiciel pour adapter la description de matériel en RTL à la structure de FPGA. De plus, cet outil effectue aussi le partitionnement de matériel en plusieurs FPGA, l'optimisation des pattes (en ajoutant des circuits additionnels) et l'encapsulation des modules réalisés dans un FPGA.

Explorer est le logiciel pour configurer, contrôler, et observer le matériel de prototypage d'Aptix (*System Explorer* et *Software Integration Station*). Ce logiciel transpose

la description partitionnée en un fichier pour configurer la plateforme de prototypage. Cette configuration consiste à établir les bonnes connexions entre les cartes. Ces connexions sont établies en utilisant FPIC.

FPGA Connector est un logiciel qui permet de développer de circuit imprimé (PCB) pour prototypage. Ce logiciel transforme la description partitionné par le *Design Pilot* au format utilisable par les outils de conception de circuit imprimé comme ORCAD ou PROTEL.

Module Verification Platform/MVP est un logiciel qui facilité l'obtention d'un environnement de co-émulation avec la plateforme System Explorer et un simulateur de modèle RTL.

2.5.4.2 Flexbench

Un autre exemple de plateforme de prototypage est *Flexbench Prototype Equipment* (Figure 8). Cette plateforme est développée par un consortium de six entreprises. Elle pour objectif un prototype avec une vitesse de plus de 100 MHz.

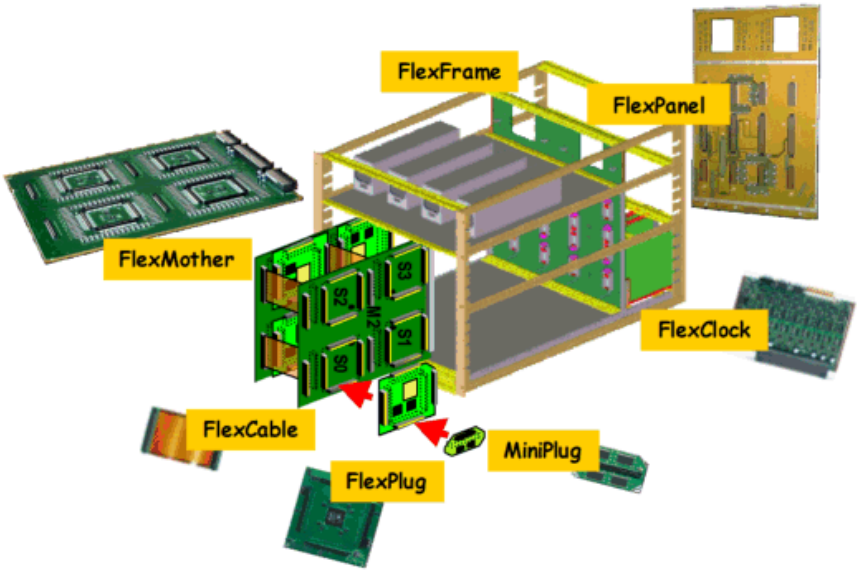


Figure 8 La plateforme de prototypage FlexBench

La plateforme Flexbench est très modulaire. Elle consiste en plusieurs types de modules : générateur d'horloge, bus PCI, FPGA, FPIC, support de mémoire (memory socket), support de processeur (processor socket), etc. Les connexions entre ces modules se

font à travers une carte mère appelée *FlexMother*. Tous ces modules sont montés dans un râtelier (rack). On peut monter aussi plusieurs types de module de déboguage. Ces modules de déboguage permettent d'observer les signaux dans chaque module en utilisant un analyseur logique.

La topologie des connexions sur la carte mère est basée sur une structure à trois dimensions. La Figure 9 illustre cette structure. Les connexions entres les modules sont configurables. Toutes les connexions ont des délais identiques de 4ns jusqu'à 5 ns. Ce qui donne une fréquence de100 MHz. Les détails de réalisation de ces connexions sont brevetés. Ils ne sont pas alors divulgués au public.

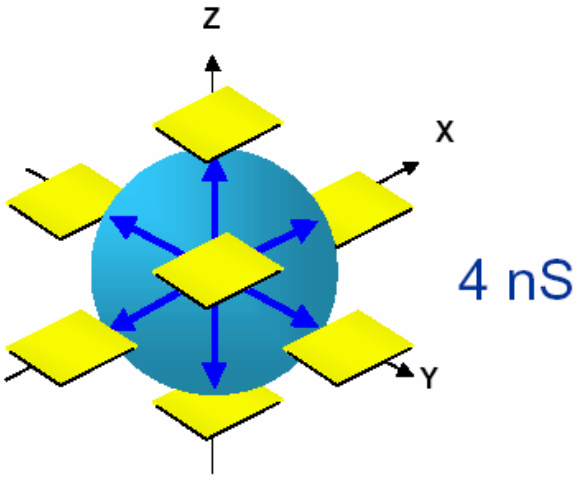


Figure 9 La topologie de FlexBench

Le prototypage en utilisant cette plateforme est commencé dès que le processus de conception arrive au niveau RTL. A partir de la description RTL de l'application, les concepteurs doivent effectuer un partitionnement pour utiliser plusieurs modules FPGA. L'outil *Certify* aide ce partitionnement. Les résultats de partitionnements sont utilisés pour faire le routage des connexions de la plateforme *FlexBench* en utilisant un outil *DiaFlex*. Si l'outil *Diaflex* n'arrive pas de faire routage, il peut produire un retour pour aider l'outil *Certify* à repartitionner l'application. Si le partitionnement est fait correctement, les concepteurs peuvent faire le placement et le routage pour chaque FPGA. Les résultats de placement, de routage, et la configuration de *FlexBench* sont téléchargés à la plateforme. La Figure 10 montre ce flot de prototypage.

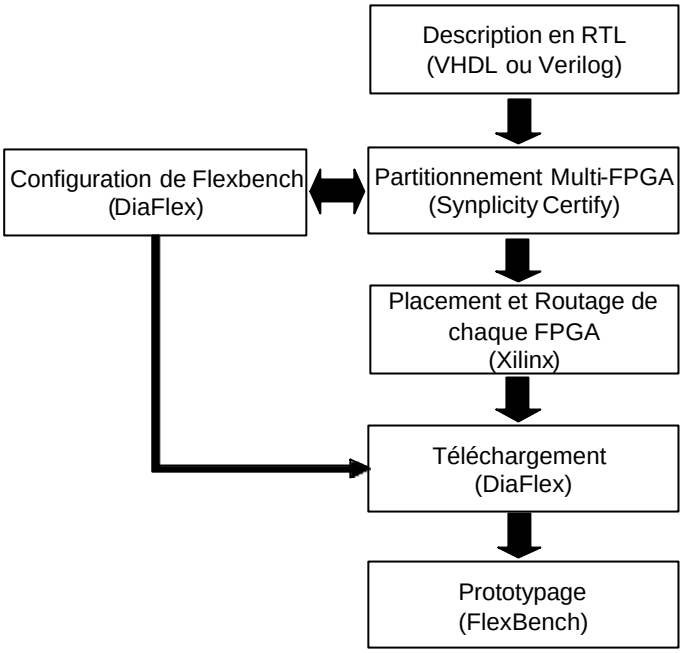


Figure 10 Flot de prototypage FlexBench

2.6 Conclusion

La vérification dans la conception de systèmes monopuces est un processus exigeant. Il existe plusieurs étapes dans la conception, et chaque étape exige une vérification avec des caractéristiques spécifiques.

Il existe aussi plusieurs techniques de vérification. Chaque technique possède ses avantages, et ses défauts. Et à chaque étape de la conception correspondent une ou plusieurs techniques de vérification.

L'émulation et le prototypage sur une plateforme reconfigurable accélèrent le processus de conception car ces deux techniques permettent de commencer le développement du logiciel plus tôt. En plus, elles sont utilisées pour tester le système dans son environnement physique. Ainsi, on réduit les possibilités de laisser des bogues dans le système. L'utilisation d'une plateforme reconfigurable permet de développer un prototype rapidement, avec moins d'efforts et pour un coût plus faible qu'un prototype spécifique.

CHAPITRE 3 : DEFINITION D'UN FLOT DE PROTOTYPAGE SUR UNE PLATEFORME RECONFIGURABLE

3.1 Introduction 40

3.2 Définition du prototypage d'un système monopuce sur une plateforme reconfigurable 40
3.2.1 Modèle de système monopuce 40
3.2.2 Modèle générique d'une plateforme de prototypage 42
3.2.3 Modèle d'une plateforme idéale pour le prototypage 43

3.3 Les plateformes reconfigurables et les flots de prototypage existants 46
3.3.1 La classification des plateformes selon l'utilisation 46
3.3.2 La classification de la plateforme reconfigurable par le nombre de domaines d'application supporté par la plateforme 46
3.3.3 La classification des plateformes reconfigurable selon la configuration ou les types de composant configurables dans la plateforme 47

3.4 Flot de prototypage simple 48
3.4.1 Allocation 48
3.4.2 Génération de code 49
3.4.3 Exemple 49

3.5 Flot de prototypage proposé 51
3.5.1 Configuration 52
3.5.2 Adaptation 54

3.6 Conclusion 56

3.1 Introduction

Le chapitre 2 a montré l'intérêt du prototypage par rapport aux autres techniques de vérification. On a aussi montré qu'une solution de prototypage basée sur une plateforme reconfigurable représentait un bon compromis entre coût et performance.

Pour réaliser un prototype à partir du modèle RTL de l'application, il est nécessaire d'avoir une méthode pour enchaîner les étapes des transformations du modèle RTL et de la plateforme. C'est le flot de prototypage.

Ce chapitre propose donc un flot de prototypage sur une plateforme reconfigurable. La première partie définit ce qu'est un flot de prototypage sur une plateforme reconfigurable et présente les modèles utilisés. La deuxième et la troisième parties présentent respectivement un flot de prototypage simple, et un flot de prototypage dans le cas réel.

3.2 Définition du prototypage d'un système monopuce sur une plateforme reconfigurable

Le prototypage d'une application sur une plateforme reconfigurable consiste à réaliser toutes les parties de l'application avec les composants disponibles sur la plateforme. Les composants matériels de l'application (processeurs, circuits spécifiques) sont réalisés avec des composants programmables (FPGA). Les composants logiciels sont réalisés par des programmes exécutés par un ou plusieurs processeurs. Selon la nature de l'application et l'architecture de la plateforme, ce processus est plus ou moins complexe.

3.2.1 Modèle de système monopuce

La Figure 11 (a) montre le modèle d'architecture générique d'un système monopuce que nous utilisons pour le prototypage. Le développement de ce modèle est basé sur le fait que ces systèmes sont représentés par un ensemble de taches. Après plusieurs étapes de raffinement, la conception atteint le niveau RTL. A ce niveau, quelques unes des tâches sont exécutées par des processeurs, et les autres tâches sont effectuées par des composants matériels spécifiques (IP). Aussi, le modèle d'un système monopuce comprend deux types de nœuds de calcul. Le premier type de nœud contient un processeur, et le logiciel exécuté par ce processeur. Ce type de nœud est appelé nœud de calcul logiciel. Le deuxième type de nœud contient un « IP ». Ce type de nœud est appelé nœud de calcul matériel.

Les données sont échangées entre les nœuds de calcul logiciel et matériel à travers un (ou plusieurs) réseau de communication, qui peut être un bus partagé, des connexions point à point, des « crossbars », un réseau sur puce ou une combinaison de ces éléments.

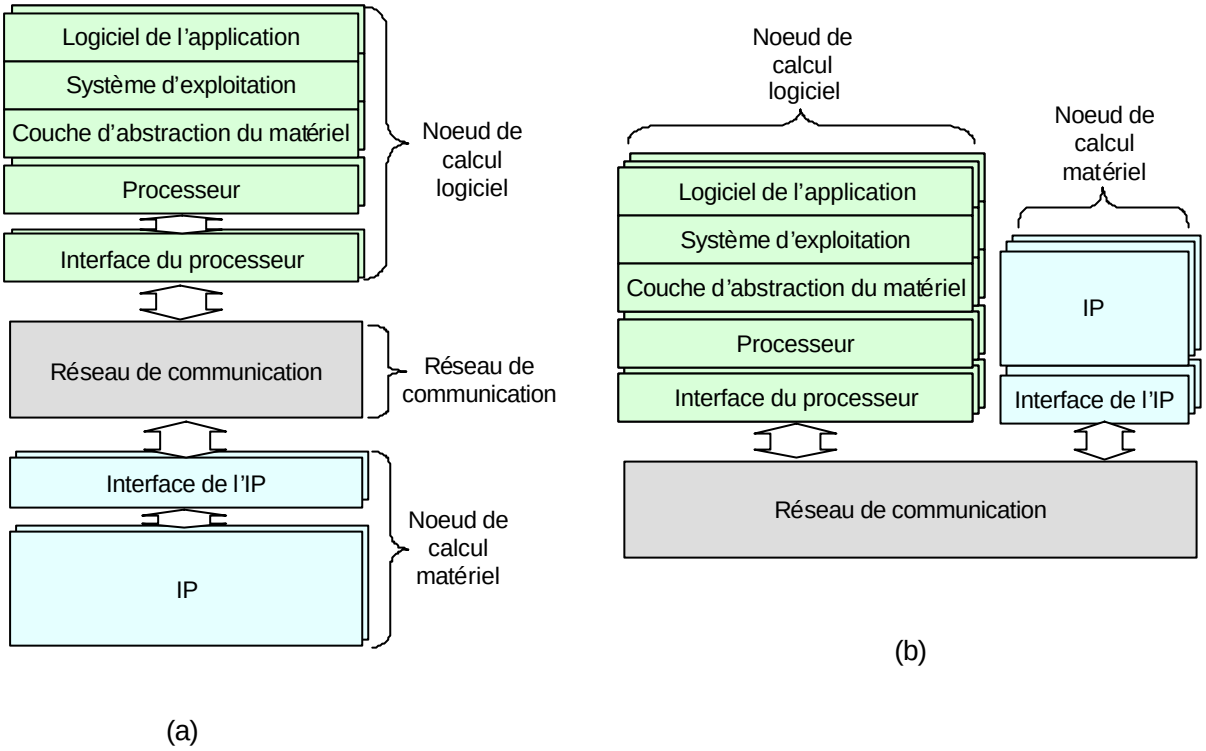


Figure 11 Modèle d'architecture d'un système monopuce

Le nœud de calcul logiciel contient le logiciel de l'application, un système d'exploitation (SE), une couche d'abstraction du matériel (en anglais : hardware abstraction layer/HAL), le processeur, et l'interface du processeur. Le logiciel de l'application, la couche d'abstraction du matériel, et le SE sont les parties logicielles de l'application. Ces logiciels sont exécutés par le processeur. Le logiciel d'application est un ensemble de tâches utilisé pour construire le comportement de l'application décrit dans la spécification fonctionnelle. Le SE et la couche d'abstraction du matériel sont les parties logicielles qui gèrent l'exécution de ces tâches et l'utilisation de ressources partagées par les tâches. Ceci inclut l'utilisation du matériel. La couche d'abstraction du matériel est séparée du SE pour permettre de porter facilement le SE et le logiciel de l'application sur différentes architectures. Quelques parties de la couche d'abstraction du matériel doivent être écrites en assembleur. Les autres parties de la couche d'abstraction du matériel et les autres programmes sont écrits en langage de

programmation de haut niveau (C ou C++). L'interface du processeur est là pour adapter l'interface du processeur au réseau de communication.

Le nœud de calcul matériel consiste en un IP et son interface. L'IP est un composant matériel qui effectue un calcul spécifique ou une tâche spécifique. L'interface de l'IP adapte l'interface physique de l'IP au réseau de communication. Ces nœuds de calculs sont décrits au niveau RTL ou au niveau portes logiques pour permettre le prototypage. La Figure 11 (b) montre une autre représentation de ce modèle que l'on utilise dans la suite de ce paragraphe.

3.2.2 Modèle générique d'une plateforme de prototypage

La plateforme de prototypage comporte des nœuds pour exécuter les logiciels, et des nœuds pour effectuer les tâches matérielles. La Figure 12 montre un modèle de plateforme pour le prototypage. Dans ce modèle, on divise la plateforme de prototypage en trois parties : les nœuds de prototypage logiciel, les nœuds de prototypage matériel, et le réseau de prototypage de communication.

Le nœud de prototypage logiciel contient au minimum un processeur et des mémoires. Le logiciel est ainsi chargé en mémoire et exécuté par le processeur. Ce noeud contient aussi l'interface pour se connecter au réseau de prototypage de communication.

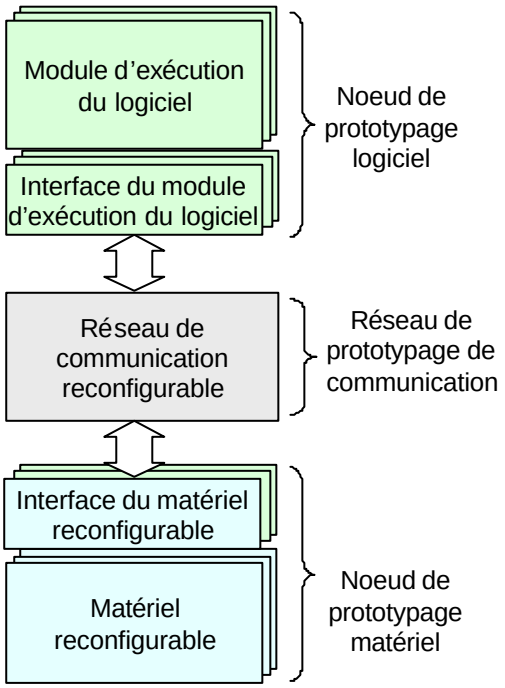


Figure 12 Modèle générique de plateforme de prototypage

Le nœud de prototypage matériel contient au moins un composant reconfigurable. La fonctionnalité d'un composants spécifique de l'application est reproduite par ce composant reconfigurable sous certaines conditions (description niveau RTL, …). Ce nœud a besoin aussi d'une interface pour se connecter au réseau de prototypage de communication.

Ces nœuds de prototypage logiciel et matériel sont connectés à travers un réseau de prototypage de communication, qui permet de reproduire un ou plusieurs schémas de communication.

Pour être capable de modéliser des architectures variées, une plateforme reconfigurable doit avoir les caractéristiques suivantes :

1. Etre modulaire, ce qui signifie que les concepteurs peuvent choisir les types et le nombre de nœuds de la plateforme.
2. Le nœud de prototypage logiciel peut remplacer tous les types de processeur et son architecture locale pour modéliser un nœud de calcul logiciel.
3. Le nœud de prototypage matériel est capable de modéliser un IP et son interface.
4. Le réseau de communication entre les nœuds peut être configuré pour réaliser divers types de communication.

3.2.3 Modèle d'une plateforme idéale pour le prototypage

Comme déjà mentionné précédemment, une plateforme idéale pour le prototypage doit être modulaire. Donc, chaque noeud est implémenté sur un circuit imprimé séparé pour avoir de la modularité. Malheureusement, les connexions entre les modules sont toujours physiquement fixées (sur un circuit imprimé). On a besoin d'un mécanisme pour utiliser ces connexions fixées permettant d'émuler les connexions entre plusieurs nœuds de prototypage logiciel/matériel et nœuds de prototypage de la communication. Dans le cas de nœuds de prototypage logiciel, ce mécanisme doit émuler des connexions entre le processeur et le réseau de communication reconfigurable. Dans le cas de nœud de prototypage matériel, ce mécanisme doit émuler des connexions entre l'IP et le réseau de communication reconfigurable. Ce mécanisme est appelé protocole de prototypage. Ce protocole consiste en des connexions fixées entre des nœuds, un adaptateur à chaque nœud pour utiliser ce protocole, et un contrôleur pour gérer l'utilisation du bus partagé.

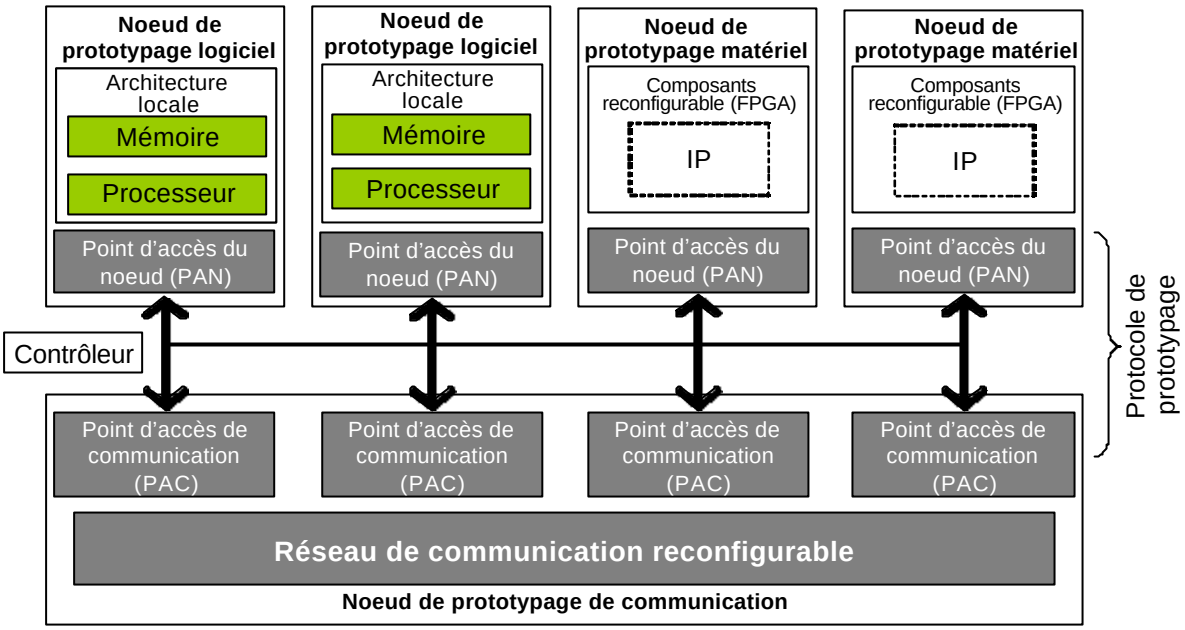


Figure 13 L'implémentation d'une plateforme idéale

La Figure 13 montre un exemple de réalisation d'une plateforme de prototypage idéale. Cette plateforme est composé de trois types de nœuds :

- Le nœud de prototypage logiciel. Il contient un processeur, et l'architecture locale de ce processeur. Il faut que le nœud permette aux concepteurs de sélectionner ou de configurer le processeur et l'architecture locale utilisée. Il contient aussi un bloc pour connecter l'interface du processeur aux composants de communication. On appelle ce bloc un point d'accès de nœud (PAN).
- Le nœud de prototypage matériel. Ce nœud contient un composant reconfigurable pour émuler l'IP. Il contient aussi un bloc pour connecter les interfaces des composant matériels aux composants de communication en utilisant le protocole de prototypage. On l'appelle aussi le point d'accès du nœud (PAN) comme pour le nœud de prototypage logiciel.
- Le nœud de prototypage de communication. Ce nœud est utilisé pour implémenter les composants de communication : des bus, des connexions point à point, des « cross bars ». Pour la réalisation de ce composant, on utilise des composants reconfigurables (FPGA). Ce nœud contient aussi des blocs point d'accès de communications (PAC) pour connecter ces composants de communication aux points d'accès des nœuds (PA N).

Les communications entre les nœuds sont implémentées en utilisant un protocole de prototypage qui consiste en des connexions fixées entre des nœuds, des adaptateurs dans chaque nœud (PAN et PAC), et un contrôleur. Pour chaque cycle :

- Le PAN encode l'information issue d'un processeur ou d'un IP et décode l'information provenant d'un PAC.
- Le PAC encode (et décode) l'information envoyée par une PAN à travers le réseau de communication.
- Les connexions physiques sont partagées par plusieurs PAC et PAN.
- Le contrôleur gère le partage des connexions physiques par les PAC et les PAN, et puis, quand toutes les communications sont effectuées, il envoie des signaux à tous les nœuds pour faire avancer d'un cycle d'horloge.

La plateforme idéale détaillée dans la Figure 13 peut aussi se représenter sous une forme plus simple et plus proche du modèle générique de la Figure 12. Cette représentation (Figure 14) fait apparaître les nœuds de prototypage logiciel comportant de la mémoire et un processeur, les nœuds de prototypage matériel comportant un composant reconfigurable, et un nœud de prototypage de communication. Les points d'accès (PAN et PAC) sont réalisés avec des composants programmable (FPGA).

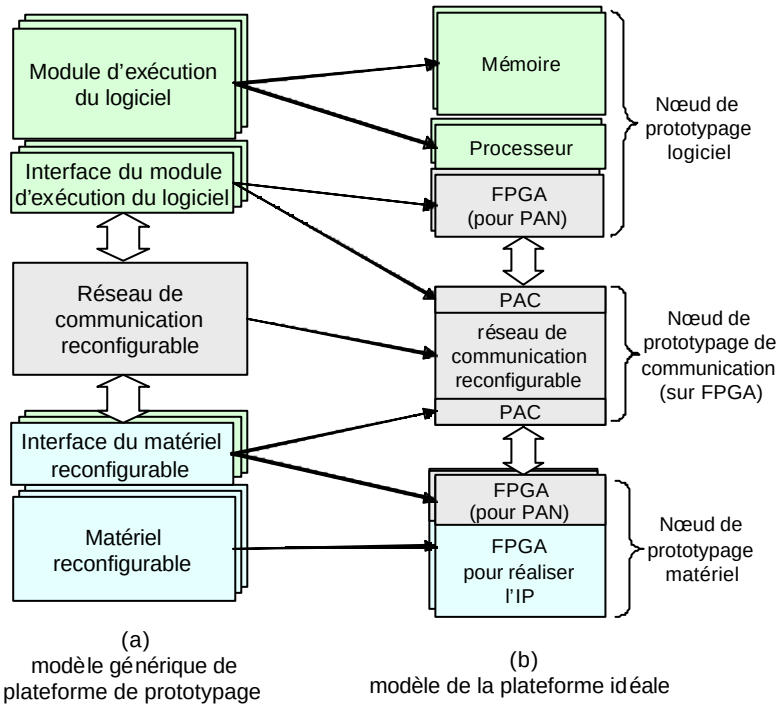


Figure 14 Modèle de notre implémentation de la plateforme idéale

3.3 Les plateformes reconfigurables et les flots de prototypage existants

Ce paragraphe décrit quelques classifications de plateformes reconfigurables et quelques flots de prototypage.

Les plateformes reconfigurables existantes peuvent être classifiées selon trois points de vues : (1) selon l'utilisation faite de la plateforme, (2) selon le nombre de domaines d'application supporté par la plateforme, (3) selon la configurabilité ou les types de composants reconfigurables dans la plateforme.

3.3.1 La classification des plateformes selon l'utilisation

Une plateforme reconfigurable peut être employée comme architecture finale du système ou comme émulateur. Si elle est employée comme architecture finale, l'exploration d'architecture du système est fortement limitée par la plateforme. Mais ce type de plateforme fournit aussi beaucoup d'autres facilités pour accélérer le processus de conception, par exemple des IP, un SE adapté et une topologie spécifique de communication. Un exemple de ce type de plateforme est la plateforme PrimeXsys de ARM [ARM01]. Cette plateforme est conçue pour des applications sans fil. Le processeur ARM 926EJ-S est à la base du noeud de prototypage logiciel. Les autres noeuds de traitement peuvent être ajoutés comme des périphériques (compteur de temps, contrôleur d'interruption, etc..), comme un IP spécifique à l'application (MPEG, DBS, etc..). La communication entre ces noeuds est basée sur six couches de bus AMBA AHB.

Comme émulateur, la plateforme configurable est employée pour valider et explorer des solutions pour l'application. Le système de prototypage proposé par M. Dorfel [DOR01] est un exemple de ce type de plateforme. L'auteur utilise le multiprocesseur basé sur VME et le PC pour exécuter les tâches logicielles. Un FPGA est utilisé pour réaliser les tâches en matériel. Des communications entre les noeuds sont exécutées par des connexions entre FPGAs et le bus VME.

3.3.2 La classification de la plateforme reconfigurables par le nombre de domaine d'application supporté par la plateforme

Nous pouvons distinguer deux classes : la plateforme configurable pour un domaine d'application spécifique, et la plateforme reconfigurable non spécifique. LOPIOM [MOS96] est un exemple de plateforme reconfigurable non spécifique. Elle est conçue pour le

prototypage des différents domaines de l'application. Des tâches de logiciel sont exécutées par un processeur Motorola MC68331. Quatre FPGA Xilinx composent les noeuds de prototypage matériels. Des communications entre les noeuds sont faites avec composant d'interconnexions programmables (FPIC) et un bus VME.

Dans le cas d'une plateforme spécifique, les composants et l'architecture sont spécialement adaptés aux spécificités des applications. Un exemple d'une telle plateforme est décrit par A. Ramanathan [RAM01]. Le noeud de prototypage logiciel supporte un StrongArm, et les noeuds de prototypage matériels contiennent deux FPGA APEX, CNA, I/Q, et un contrôleur Ethernet. La communication est obtenue à l'aide d'un bus standard (données, adresses, signaux de contrôle), d'un lien Ethernet et de quelques signaux de commande dédiés.

3.3.3 La classification des plateformes reconfigurables selon la configuration ou les types de composants configurables dans la plateforme

On a trois types de plateforme basées sur ce critère. Une plateforme avec des composants matériels reconfigurables, une plateforme avec des composants programmables pour exécuter du logiciel, et une plateforme avec des connexions reconfigurables.

Un plateforme avec matériel reconfigurable utilise différents types varies de composants reconfigurable : FPGA, CPLD, PLA, … . Elle permet alors de réaliser de nombreux modules matériels. Il existe aussi des plateformes avec des composants matériels reconfigurables moins classiques, dont la granularité moins fine, comme par exemple FPAA (*Field programmable ALU array*, une composant reconfigurable composé par un ensemble d'unités arithmétiques et logiques) [HAR97b], un autre exemple est la plateforme décrite en [SAS03d]. Un exemple de ce type de plateforme est décrit par K. Adaeis [ADA98]. Cette plateforme possède quatre FPGA. Ces FPGA sont connectés par quelques fils, dont certains sont partagés par plusieurs FPGA..

Une plateforme avec des composants programmables est composée d'un ou plusieurs processeurs. Cette plateforme peut aussi avoir quelques modules matériels mais ils ne sont pas reconfigurables. Un exemple est WinSystems Embedded PC [WINSY]. Cette plateforme utilise des processeurs utilisés dans le PC (Pentium d'Intel, AMD, …). Le système d'exploitation est mis dans un composant nommé *Disc-On-Chip* (DoC). Ce composant agit comme un disque dur.

Le troisième type de plateforme possède des connexions configurables. La réalisation des connexions reconfigurables est fait soit en utilisant des composants reconfigurables comme FPIC, soit en utilisant d'une architecture spécifique. Une plateforme APTIX et la plateforme Flexbench présentées dans le paragraphe 2.5.4 sont des exemples de ce type de plateforme. Evidemment, une plateforme peut correspondre à sur plusieurs classes de reconfigurabilité si elles ont plusieurs composants configurables ou programmables.

3.4 Flot de prototypage simple

Le flot de prototypage est très simple si l'architecture de la plateforme est très proche de l'architecture de l'application. Dans ce cas, le flot de prototypage se limite à une étape d'allocation («assignment» en anglais) suivie d'une étape de génération de code, compilation et synthèse («targeting» en anglais) (Figure 15).

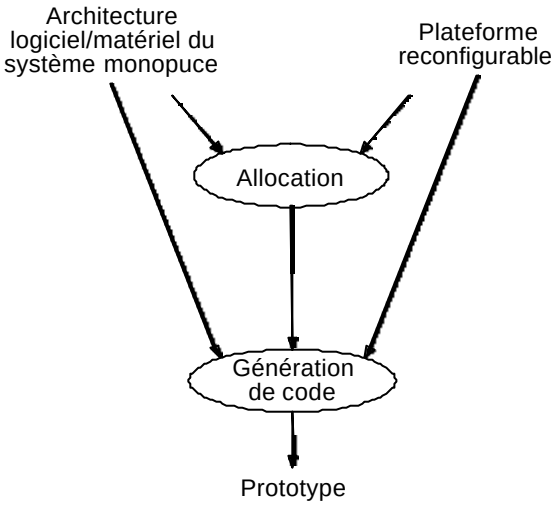


Figure 15 Flot de prototypage simple

3.4.1 Allocation

Dans l'étape d'allocation, les concepteurs associent chaque partie de l'architecture à un nœud de prototypage de la plateforme. Ces associations indiquent sur quelles parties de la plateforme reconfigurable sont réalisées les parties de l'architecture de l'application. Le résultat de cette étape est un tableau d'association. Ce tableau guide toute la suite du processus de prototypage.

Les décisions prises pour faire ces associations sont basées sur les objectifs du prototypage et les contraintes imposées par la plateforme. Ce n'est pas du partitionnement logiciel/matériel car l'application est déjà décrite à un bas niveau d'abstraction.

3.4.2 Génération de code

La génération de code (« ciblage») est la dernière étape dans le flot de prototypage. On le fait si chaque partie de l'application a une correspondance directe avec les composants de la plateforme. Cette étape consiste en processus standard tels que la compilation et l'édition de lien des logiciels, la synthèse logique, le placement sur FPGA, et le routage.

3.4.3 Exemple

La Figure 16 donne un exemple d'allocation du prototypage d'une application sur une plateforme reconfigurable. L'application est décrite en plusieurs couches, avec le code d'application, le système d'exploitation et la couche d'abstraction du matériel s'exécutant sur un processeur. Ce processeur est connecté au réseau de communication à travers un composant d'adaptation (interface du processeur). Le composant matériel spécifique est lui aussi connecté au réseau par un adaptateur d'IP.

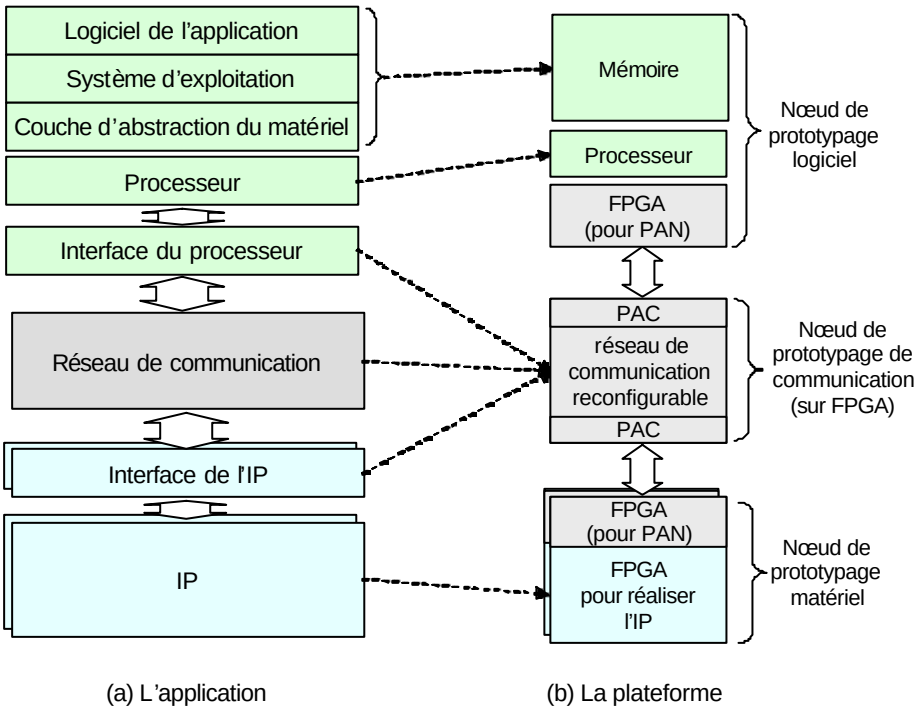


Figure 16 Exemple de l'allocation

Supposons que l'on veuille prototyper cette application sur la plateforme idéale présentée dans le paragraphe 3.2.3. et que les interfaces entre les nœuds de prototypage logiciel/matériel et le nœud de prototypage de communication sont mis en œuvre par des FPGA. L'étape d'allocation est ici simple. Elle consiste à dire que les programmes sont placés dans la mémoire programme du processeur, le processeur de l'application est émulé par le processeur de la plateforme, l'adaptateur du processeur, l'adaptateur du IP, et le réseau de communication sont réalisés par le réseau de communication de l'application. Le composant IP est réalisé par le FPGA du nœud de prototypage matériel.

L'étape de génération de code se décompose en plusieurs parties (Figure 17) :

- Pour la partie logicielle : la génération du code consiste à ajouter un module de déboguage (si nécessaire), puis à faire la compilation et l'édition de lien.
- Pour le processeur, on choisir un processeur pour exécuter la partie logicielle. Le PAN de ce nœud de prototypage logiciel est alors configuré pour encoder (et décoder) les signaux des entrées/sorties du processeur. Ce PAN est un module VHDL paramétrable. On configure le PAN, suivi de la synthèse et du placement et routage avant le chargement sur FPGA de ce nœud.
- Les interfaces des IP et des processeurs et aussi le réseau de communication sont réalisés par le réseau de communication réconfigurable.
- Les IPs sont implémentés sur un ou plusieurs FPGA (dans les nœuds de prototypage matériel). Pour faire le partitionnement de ces IP, on doit d'abord faire d'une estimation des tailles des IPs. Basé sur les tailles estimées, on décide le nombre de nœuds de prototypages matériel dont on a besoin et ensuite, on partitionne les IP sur ces nœuds. Pour chaque nœud, on doit ajouter un PAN, et éventuellement un module de déboguage si c'est nécessaire. Le module de plus haut niveau est ensuite ajouté à chaque nœud pour encapsuler tous les modules (Les IP, PAN, et les modules de déboguages). Enfin, on fait la synthèse et le placement-routage pour chaque nœud.

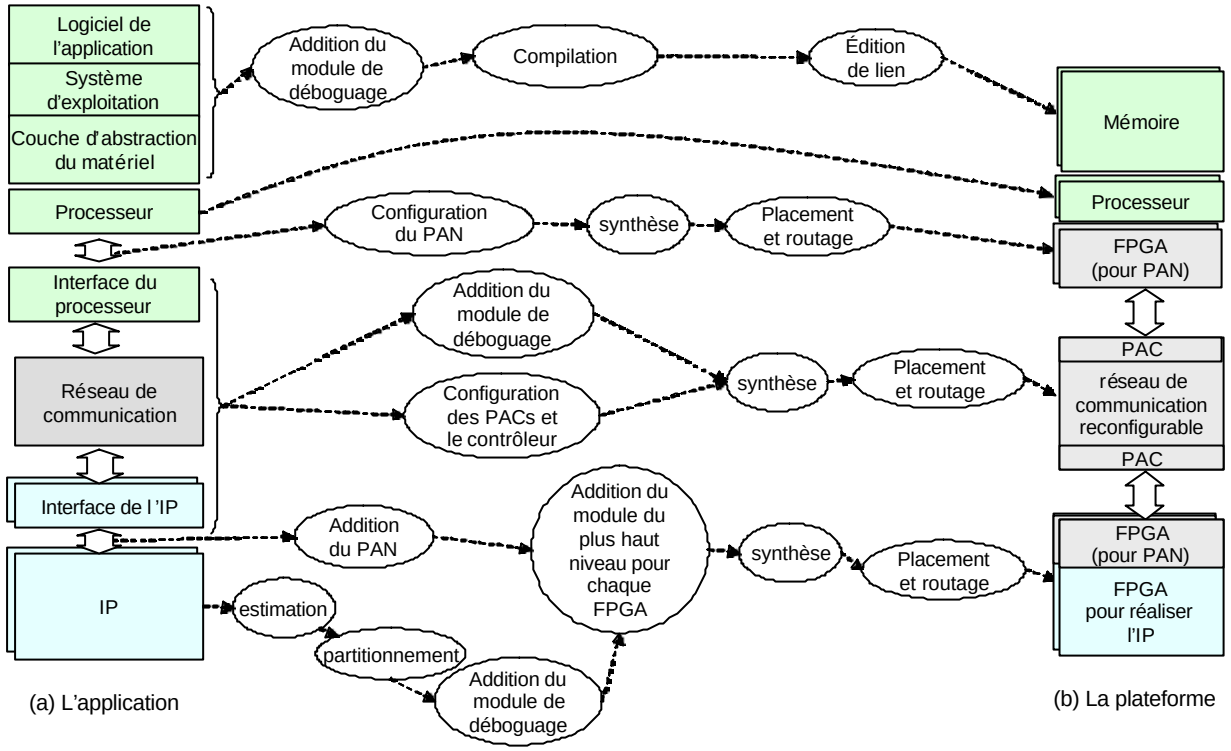


Figure 17 Génération du code

3.5 Flot de prototypage proposé

Malheureusement, le flot de prototypage simple présenté précédemment s'applique quand l'architecture de la plateforme est proche de l'architecture de l'application, ce qui est rarement le cas dans la réalité. De nombreuses différences peuvent exister : le réseau de communication de la plateforme fixé n'est pas celui de l'application, les processeurs ne sont pas du même type, la carte mémoire est différent et dans certains cas les programmes de démarrage, le traitement des exceptions et la gestion des interruptions sont à reconsidérer.

Dans ce cas, pour prototyper l'application sur la plateforme reconfigurable, il est nécessaire de réorganiser la plateforme et éventuellement d'apporter quelques modifications à l'architecture de l'application.

Ainsi nous proposons un flot avec quatre étapes (Figure 18) : (1) **allouer** chaque partie de l'application à la plateforme, (2) **configurer** la plateforme pour qu'elle convienne à l'architecture de l'application, (3) **adapter** l'architecture de l'application si nécessaire, (4) **générer le code** pour les nœuds de prototypage de la plateforme.

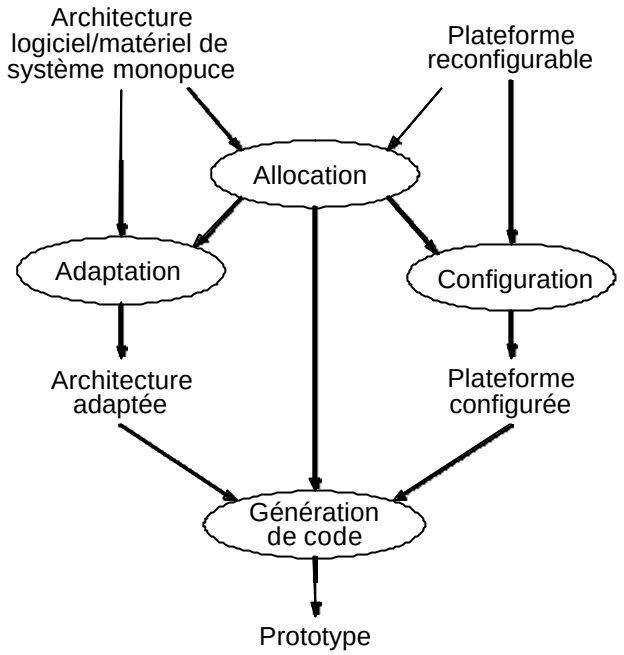


Figure 18 Flot de prototypage

Dans l'étape d'allocation dans le flot de prototypage réel, les concepteurs doivent décider des parties de l'application qui seront vraiment implémentées, et choisir les parties dont on doit émuler les fonctionnalités différemment. Dans cette étape, on décide aussi comment on fait l'adaptation et la configuration. Ces décisions dépendent des objectifs du prototypage.

3.5.1 Configuration

Après l'étape d'allocation, les deux étapes suivantes sont la configuration et l'adaptation. On peut définir la configuration comme la réorganisation de la plateforme reconfigurable. On modifie la plateforme reconfigurable pour que son architecture soit plus proche de celle de l'architecture de l'application.

La Figure 19 montre un exemple de configuration. Dans cet exemple, on veut prototyper une architecture (Figure 19.a) qui contient un processeur, un IP, et un bus X. La plateforme disponible consiste en un nœud de prototypage logiciel contenant le même type de processeur, et un nœud de prototypage matériel contenant un FPGA qui permet de réaliser l'IP. Malheureusement, la communication entre le processeur et le FPGA est différente (bus Y) et fixée. On doit alors implémenter le bus X dans le FPGA. Afin de le faire, on a besoin d'un convertisseur pour cacher le bus Y.

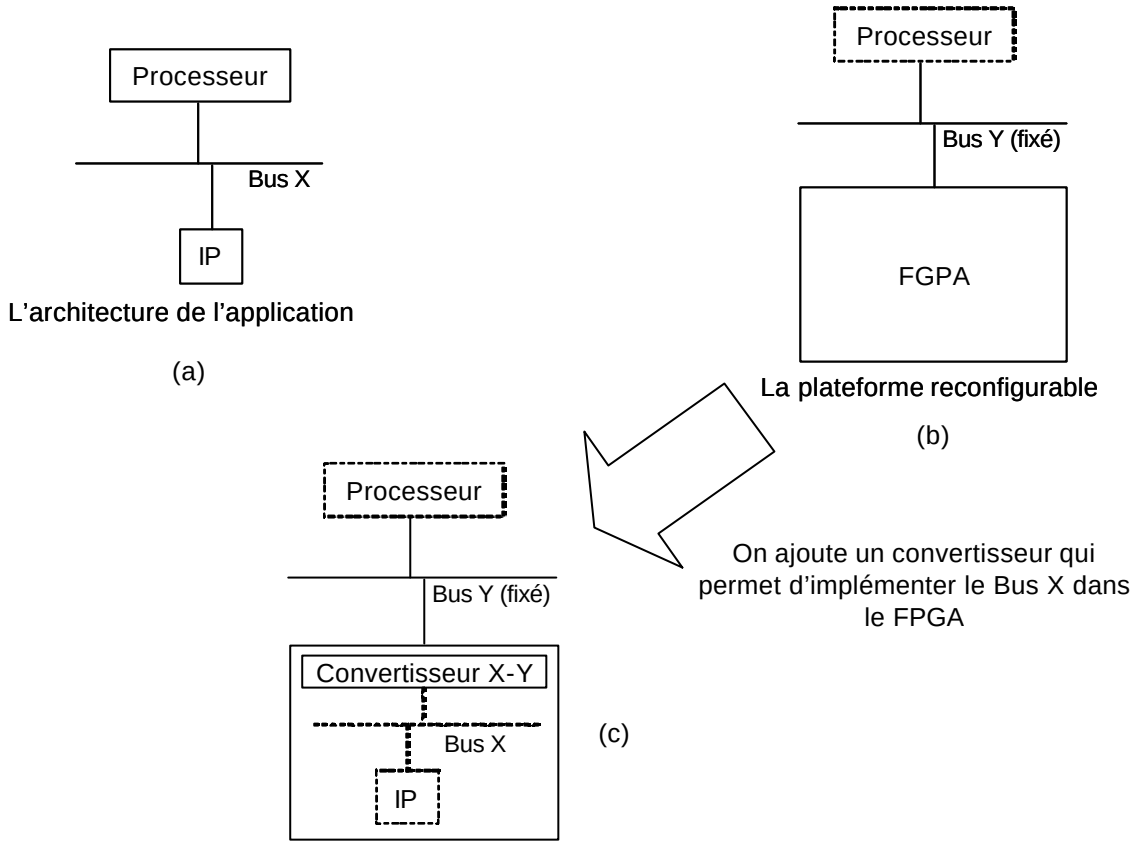


Figure 19 Exemple de configuration

On peut appliquer cet exemple à partir de la structure en couche de l'application et de la plateforme, en reprenant le principe de la Figure 16. Sur la Figure 20, la plateforme possède un réseau de communication fixé pour connecter le nœud de prototypage matériel au nœud de prototypage logiciel.

Si l'objectif du prototypage suppose de réaliser le bus X, alors il apparaît évident de placer le bus X ainsi que tous le composants d'interface de l'application dans le nœud de prototypage matériel. Il est donc nécessaire d'ajouter une couche de conversion entre le réseau Y et le bus X. Le réseau fixé devient transparent pour l'IP grâce à ce convertisseur en terme de protocole entre l'IP vers le processeur. Ceci n'est pas vrai pour les communications du processeur vers l'IP, puisque le nœud de prototypage logiciel est connecté au réseau Y.

Dans ce cas là, la configuration consiste à développer le convertisseur entre le réseau fixé de la plateforme et l'interface du processeur. Ce convertisseur est réutilisable si on utilise la même plateforme.

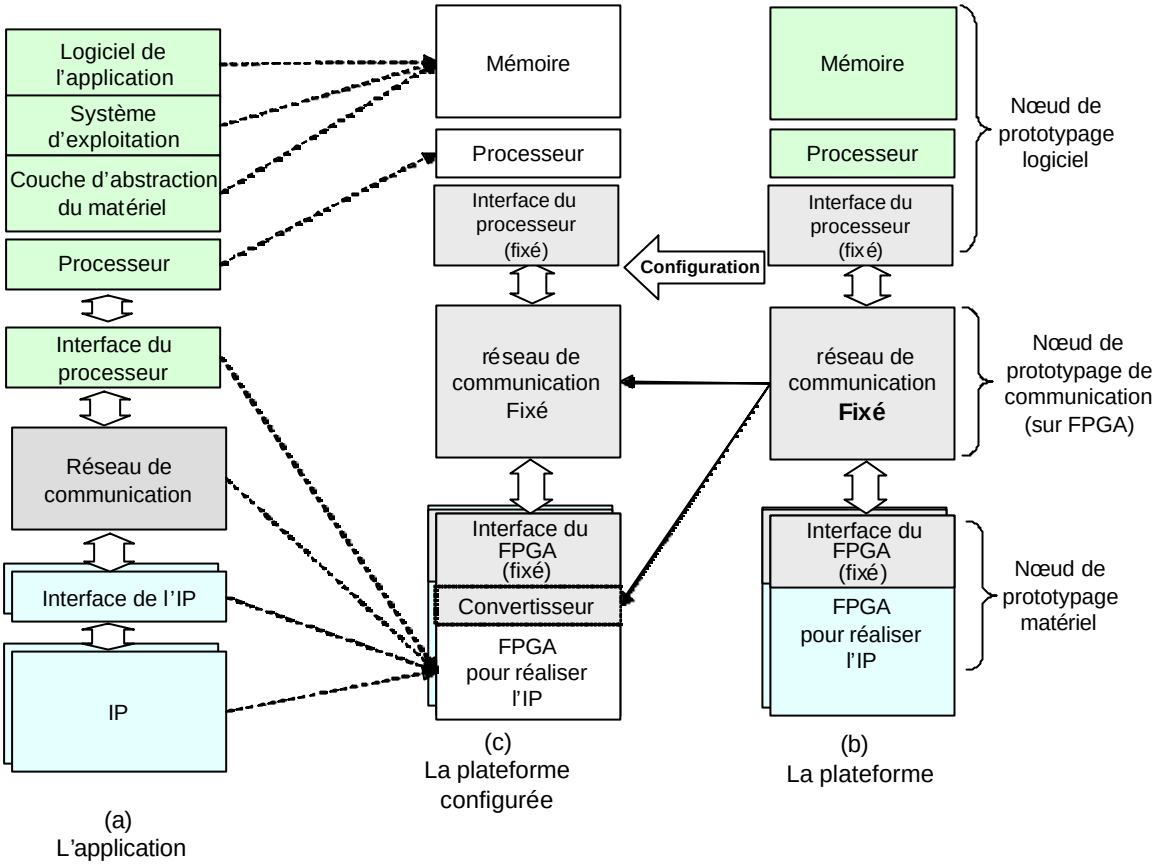


Figure 20 Configuration

3.5.2 Adaptation

L'adaptation consiste à modifier l'application pour satisfaire aux caractéristiques de la plateforme reconfigurable. Cette étape est effectuée si la plateforme ne peut pas être configurée pour s'adapter aux besoins de l'application. Par exemple si la plateforme a un plan mémoire fixé, la couche d'abstraction du matériel et le système d'exploitation doivent être modifiés.

La Figure 21 montre un exemple d'adaptation dans lequel la plateforme reconfigurable possède un plan d'adresses fixé différent de celui de l'application. Dans ce cas, les concepteurs modifient le plan mémoire de l'application. Cette modification correspond à l'adaptation de l'architecture de l'application à la plateforme.

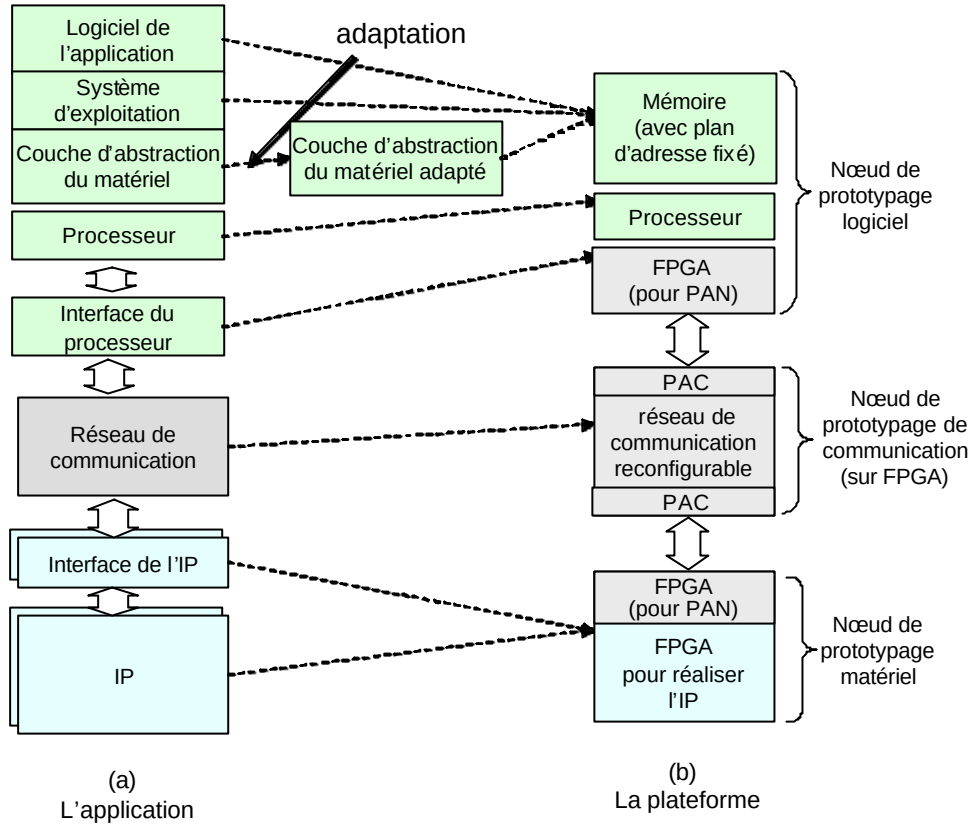


Figure 21 Adaptation

L'adaptation est une étape très laborieuse et complexe. Dans cette étape, les concepteurs doivent modifier l'architecture de l'application qui est déjà décrite à un bas niveau d'abstraction. Cette tâche est difficile, les concepteurs ont besoin de beaucoup de connaissances sur l'application, l'architecture, et la description de bas niveau. Une automatisation sur cette étape serait donc très utile.

La configuration est préférée à l'adaptation, si les étapes sont toutes les deux possibles pour résoudre une incompatibilité, ce qui évite de modifier le code de l'application. On peut alors vérifier toutes les parties du code de l'application car il est inchangé. De plus, on préfère ne pas modifier le code de l'application, car c'est souvent difficile à faire.

Après la configuration et l'adaptation, l'étape de génération de code correspond à la synthèse logique, le placement et routage pour réaliser les composants matériels, le réseau de communication et l'interface de conversion. Pour la partie logicielle, le processus est identique à celui du prototypage simple : compilation et édition de lien et chargement.

3.6 Conclusion

Nous avons présenté dans ce chapitre un flot de prototypage simple et un flot de prototypage réel. Ce dernier est le plus utilisé car généralement, l'architecture de la plateforme ne correspond pas exactement à celle de l'application, et la plateforme possède des contraintes. Ce qui oblige les concepteurs à reconfigurer la plateforme et à adapter l'application pour réaliser son prototype.

L'étape d'allocation est faite manuellement car elle nécessite des prises de décisions que seuls les concepteurs peuvent faire. De plus, cette étape fait appel à la créativité.

La configuration dépend fortement de la plateforme utilisée et de ces possibilités de reconfigurabilité.

L'adaptation est une étape qui est difficile à faire, car les concepteurs doivent travailler avec un code de bas niveau. L'automatisation est donc très utile pour réduire le temps de développement du prototypage.

La génération du code consiste en des processus standard tels que la compilation, l'édition de lien, la synthèse logique, et le routage sur FPGA. Ces processus sont déjà automatiques.

Parmi les étapes de prototypage, l'adaptation est l'étape la plus délicate. Donc, son automatisation serait très utile et permettait un gain de temps important.

CHAPITRE 4 : APPLICATIONS

4.1 Introduction 58

4.2 Plateforme ARM Integrator 58
4.2.1 La carte mère de la plateforme ARM Integrator 59
4.2.2 La carte processeur de la plateforme ARM Integrator 60
4.2.3 La carte FPGA de la plateforme ARM Integrator 60
4.2.4 Le Bus AMBA 61
4.2.5 Outil de déboguage : Multi-ICE 61
4.2.6 Logiciel de développement et de déboguage 62
4.2.7 Les caractéristiques de la plateforme ARM Integrator 63

4.3 Prototypage du VDSL sur la plateforme avec un réseau de communication fixé 64
4.3.1 Description de l'application 64
4.3.2 Spécification de l'application 64
4.3.3 Architecture logicielle matérielle 65
4.3.4 Objectif du prototypage et contraintes 65
4.3.5 Allocation 66
4.3.6 Configuration 67
4.3.7 Adaptation 69
4.3.8 Génération du code 69
4.3.9 Résultats 70

4.4 Prototypage du DivX à l'aide d'une phase d'adaptation automatique 71
4.4.1 Description de l'application 71
4.4.2 Objectif du prototypage et contraints 72
4.4.3 La spécification et l'architecture RTL 73
4.4.4 Allocation 76
4.4.5 Configuration 77
4.4.6 Adaptation 77
4.4.7 Génération du code 81
4.4.8 Résultats et analyse 81

4.1 Introduction

Dans le chapitre précédent, nous avons proposé un flot de prototypage en utilisant une plateforme reconfigurable. Ce flot nous permet d'obtenir un prototype même si l'architecture de la plateforme est différente de celle de l'application. Ce flot consiste en quatre étapes : allocation, configuration, adaptation, et génération de code.

Ce chapitre présente des exemples d'utilisation des ces techniques. Une plateforme ARM Integrator est utilisée comme plateforme de prototypage. Après avoir décrit la plateforme ARM Integrator, on montrera, à travers le prototypage d'une application VDSL et d'une application DivX les différentes phases du flot de prototypage et l'effort à faire pour concevoir un prototype.

4.2 Plateforme ARM Integrator

La plateforme ARM Integrator est une plateforme matérielle conçue par ARM Holding Plc pour développer une application sur des processeurs ARM autour d'un bus AMBA. Cette plateforme contient trois types de carte : une carte mère (nommée « ARM Integrator AP »), une carte processeur (nommée « core module »), et une carte FPGA (nommée « logic module »). Sur la carte mère, on peut monter cinq autres cartes (des cartes processeurs ou des cartes FPGA), mais on ne peut pas monter plus de quatre cartes d'un même type (quatre cartes processeur ou quatre cartes de FPGA). La Figure 22 montre une version simplifiée de cette plateforme avec trois cartes processeurs et deux cartes FPGA.

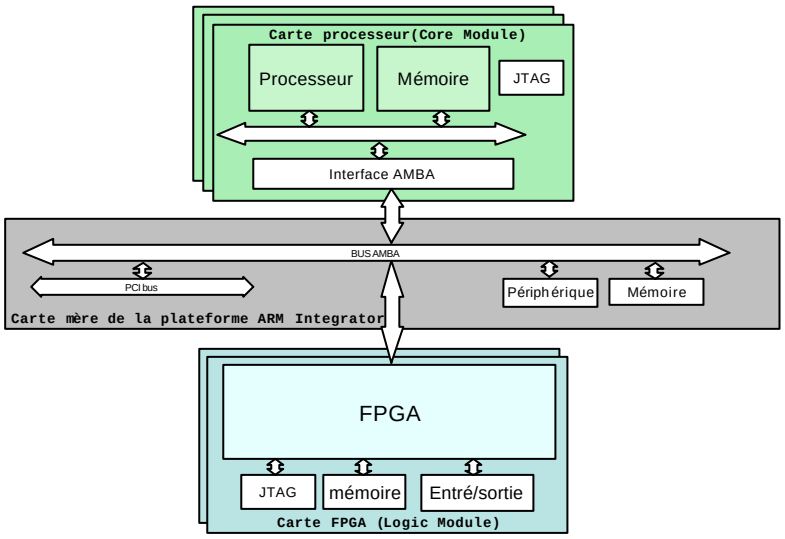


Figure 22 Plateforme ARM Integrator avec 3 cartes processeur et 2 cartes FPGA

4.2.1 La carte mère de plateforme ARM Integrator

La carte mère de la plateforme ARM Integrator [ARM99b] contient un bus AMBA, un bus PCI, quelques périphériques et de la mémoire. Le bus AMBA est découpé en deux parties (AHB et APB) connectées par un pont (en anglais : bridge). Le bus AMBA AHB connecte les sous-systèmes principaux : cartes du processeur, carte FPGA, bus PCI, et la mémoire. Le bus AMBA APB connecte les autres périphériques. Le bus PCI permet de connecter trois cartes PCI et une extension de bus PCI. Les mémoires sur la carte mère consistent en une mémoire flash, une SRAM, et une ROM. Ces mémoires sont connectées à un bus AMBA à travers un contrôleur de mémoire statique. On peut aussi connecter une mémoire additionnelle. Les périphériques connectés au bus AMBA APB sont : un compteur de temps (*timer*), une horloge temps réel (*RTC*), des entrés/sorties (*GPIO*), deux liaisons série (*UART*), un contrôleur clavier et une souris, un écran, un contrôleur d'interruptions, et quelques registres de contrôle. La Figure 23 donne une vue détaillée de la carte mère de la plateforme ARM Integrator.

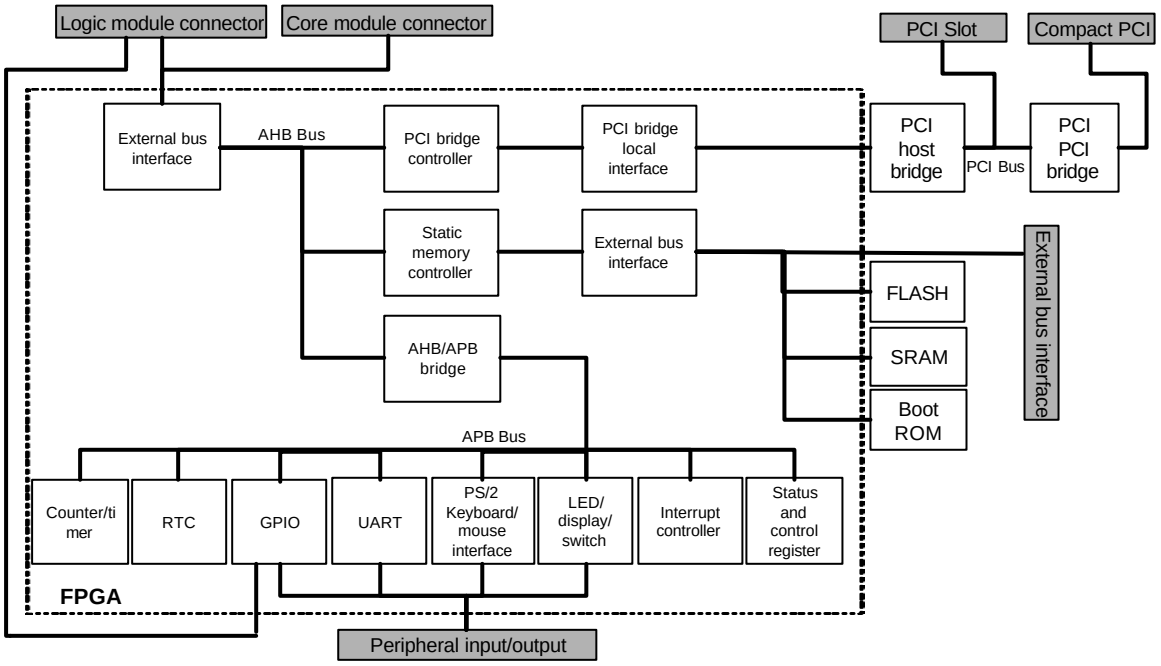


Figure 23 La carte mère

Toutes les cartes (processeur ou FPGA) peuvent accéder à tous les sous-systèmes de la carte mère. Il existe un seul plan mémoire. Ce plan mémoire comprend aussi les mémoires locales à chacune des cartes, qui sont donc partagées.

4.2.2 La carte processeur de la plateforme ARM Integrator

La carte processeur [ARM99d] contient principalement un processeur, des mémoires, et une interface du bus AMBA (Figure 24). On peut choisir le type du processeur ARM (ARM 7 et ARM 9). Il y a deux mémoires connectées au bus mémoire du processeur : SSRAM et SDRAM. L'interface du bus AMBA connecte le bus mémoire du processeur au bus AMBA sur la carte mère. On trouve aussi un contrôleur de remise à zéro, un générateur de signal d'horloge, et un module JTAG pour le déboguage.

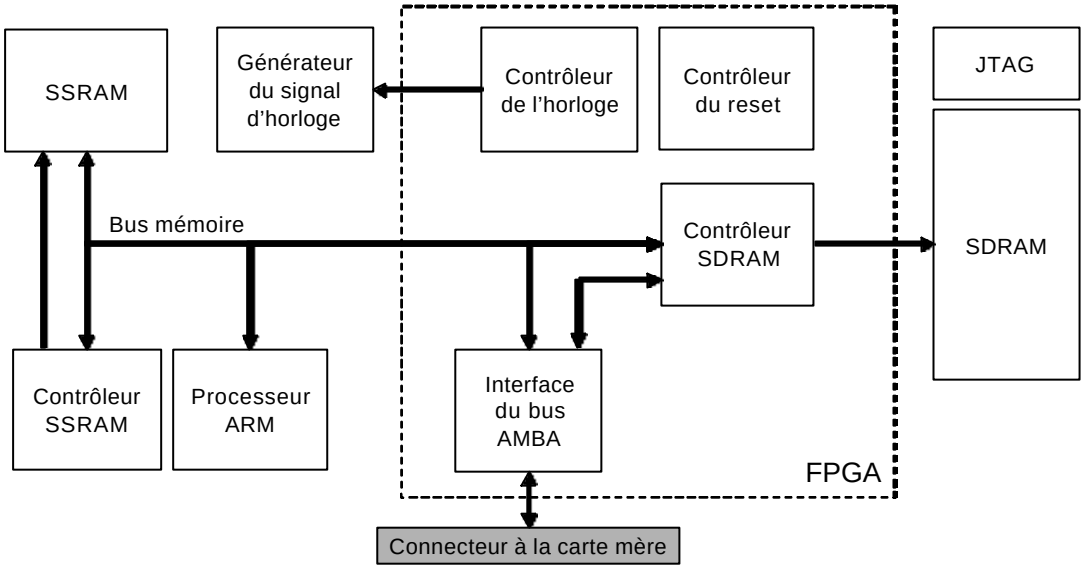


Figure 24 La carte processeur

4.2.3 La carte FPGA de la plateforme ARM Integrator

La carte FPGA [ARM99c] contient un FPGA, une interface JTAG et quelques périphériques (Figure 25). Le FPGA est connecté à tous les autres modules. Ce FPGA est configuré à partir d'ordinateur ou à partir d'une image téléchargée sur une mémoire flash. La mémoire SSRAM est connectée directement au FPGA sans décodeur adresses ou contrôleur mémoire. On doit alors réaliser ce décodeur adresses sur le FPGA. Des périphériques d'entrées/sorties sont connectés au FPGA : diodes électroluminescentes, interrupteurs, et un bouton poussoir. Le connecteur à la carte mère est relié directement au FPGA. On doit alors écrire une interface du bus AMBA pour connecter le matériel réalisé sur le FPGA à la carte mère.

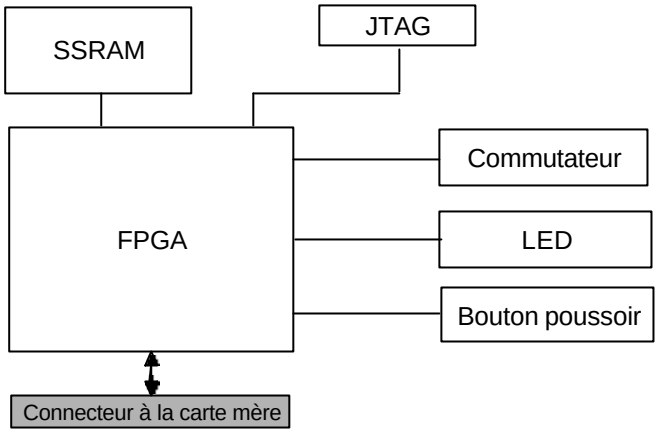


Figure 25 La carte FPGA

4.2.4 Le Bus AMBA

Les communications sont principalement réalisées par le bus AMBA [ARM99a]. Ce bus a été conçu par ARM pour les systèmes monopuces. Il existe en fait trois types de bus : AHB, ASB, APB. Le bus AHB accepte plusieurs maîtres, a des performances élevées et permet des transferts de données par paquets (mode burst). Le bus ASB possède les mêmes caractéristiques mais ne permet pas les transferts par paquets. Le bus APB est plus simple et moins performant. Il n'autorise qu'un seul maître, mais il consomme moins d'énergie (« low power »). [ARM99a] décrit le spécification des bus AMBA en détail. La Figure 26 illustre une configuration typique d'un bus AMBA.

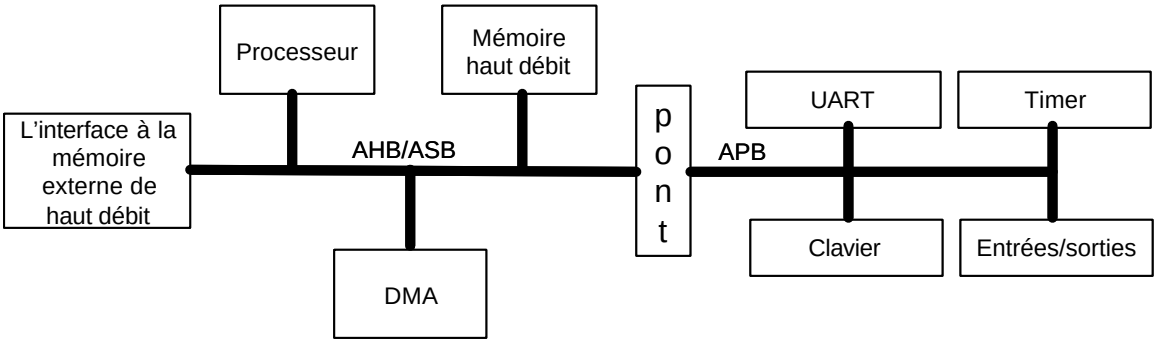


Figure 26 La configuration typique d'une bus AMBA

4.2.5 Outil de déboguage : Multi-ICE

Multi ICE [ARM99f] est un outil qui permet de connecter plusieurs logiciels de déboguage aux composants de la plateforme. On peut ainsi déboguer la partie logicielle de

chacun des processeurs dans une même exécution. Pour déboguer la partie matérielle (dans le FPGA), il faut ajouter une interface JTAG. Multi ICE est aussi utilisé pour télécharger des programmes en mémoire sur les cartes (programmation des FPGA par exemple). La Figure 27 illustre l'utilisation de multi-ICE.

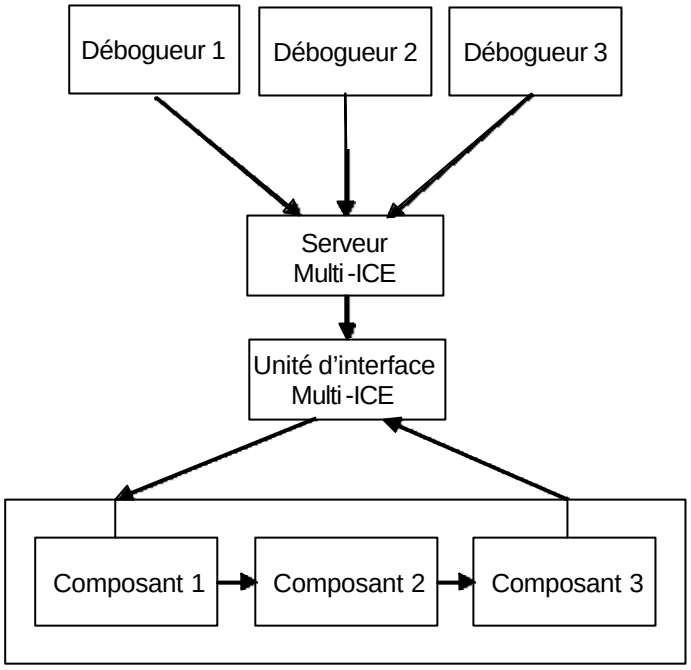


Figure 27 Multi-ICE

4.2.6 Logiciel de développement et de déboguage

Deux types de logiciels sont nécessaires pour utiliser la plateforme : les logiciels de développement et les logiciels de déboguage. Pour le développement, nous avons un compilateur (armcc pour compiler un code C, armcpp pour compile un code C++, armasm pour langage assembleur, tcc pour compiler un code C en mode 16 bit/thumb, tcpp pour un code C++ en 16 bit), un éditeur du lien (armlink), et une interface graphique pour gérer un projet (code warrior).

Pour le déboguage, nous avons un logiciel de déboguage [ARM99e] (AXD, un débogueur qui peut être connecté à la plateforme, avec un simulateur), et deux simulateurs (armsd, et ARMulator). La connexion entre le déboguer et la plateforme est à travers du Multi-ICE. On peut lancer plusieurs débogueurs connectés à plusieurs processeurs sur la plateforme.

4.2.7 Les caractéristiques de la plateforme ARM Integrator

Nous avons vu que la plateforme ARM Integrator est conçue avec trois types de cartes : carte mère, carte FPGA et carte processeur. Ces trois cartes correspondent aux trois nœuds dans le modèle de la plateforme de prototypage (paragraphe 3.2.2). Dans la plateforme, la carte processeur représente le nœud de prototypage logiciel, la carte FPGA représente le nœud de prototypage matériel, et la carte mère représente le nœud de prototypage de communication.

Contrairement à la plateforme de prototypage idéale, on trouve beaucoup de limitations dans cette plateforme. Les limitations principales sont sur la partie communication, le contrôleur d'interruptions, et sur le plan de mémoire.

Sur cette plateforme, la communication entre les cartes est fixée au bus AMBA ASB. Cette limitation existe sur la carte mère, mais aussi sur la carte processeur et sur la carte FPGA. Sur la carte processeur, les entrées/sorties passent par l'interface du bus AMBA. Sur la carte FPGA, les broches du FPGA sont directement connectées au bus AMBA de la carte mère.

Le plan mémoire globale de la plateforme permet à tous les nœuds de prototypage d'accéder à tous les composants mémoire, et plus généralement à l'ensemble des ressources de toutes les cartes. Ce plan mémoire est figé. Par exemple, l'espace d'adressage des cartes FPGA est fixé de 0xC000 0000 à 0xFFFF FFFF et l'adresses des mémoire partagées sont fixée entre 0x8000 0000 jusqu'à 0xBFFF FFFF.

Le contrôleur d'interruptions est fixé et centralisé. De plus, une seule broche du FPGA permet d'envoyer une interruption aux quatre processeurs. C'est une limitation importante pour le développement d'un prototype.

Malgré ces limitations, la plateforme ARM Integrator est un bon outil pour déboguer le logiciel, surtout, si l'application utilise un bus AMBA. Mais, il n'est pas facile de déboguer le matériel avec cette plateforme. En effet, cette plateforme est conçue pour le développement du logiciel et du matériel connecté au bus AMBA. Elle n'est pas conçue pour développer d'autres architectures. Dans la suite de ce chapitre, nous allons décrire nos expérimentations en utilisant cette plateforme.

4.3 Prototypage du VDSL sur la plateforme avec un réseau de communication fixé

4.3.1 Description de l'application

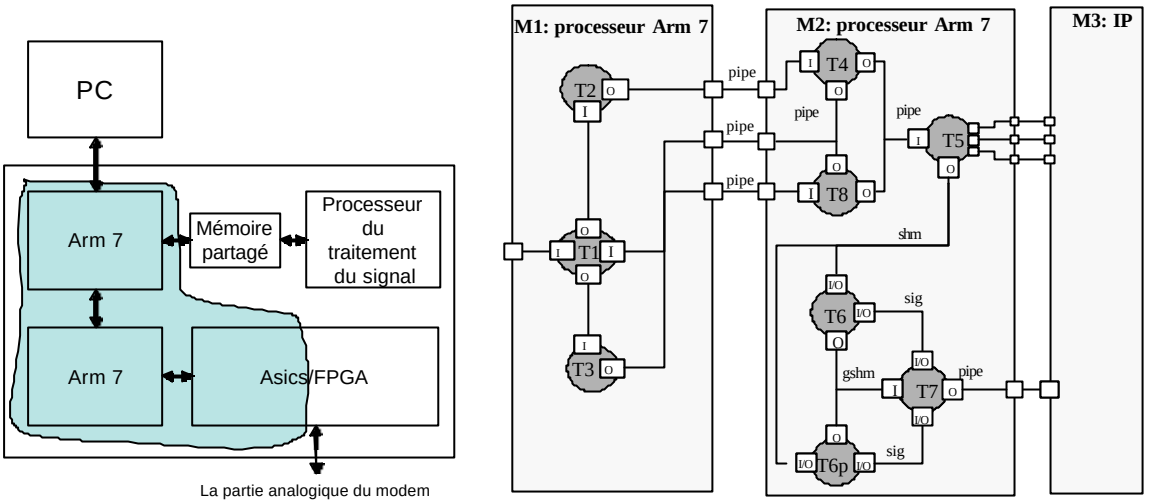
DSL (Digital Subscriber Line) est une application pour utiliser une ligne de communication téléphonique (analogique) pour transférer un signal numérique entre deux appareils. L'utilisation d'un signal numérique permet le transfert de données de haut débit et permet d'augmenter la fiabilité. VDSL est une version du DSL. Le VDSL possède une très haute performance mais suppose une distance réduite entre deux appareils.

La version du VDSL utilisée dans ce chapitre vient de [MES00]. Cette version utilise le codage DMT (Discret Multi Tone) qui découpe la bande de fréquences initiale en sous-bandes de transmission simultanée. Le « Zipper » découpe la bande de fréquence en 2048 sous-bandes qui sont allouées dynamiquement par logiciel, à différents utilisateurs.

4.3.2 Spécification de l'application

Pour des raisons de disponibilité du code source, seule une partie du système a été réalisée. La spécification de départ est donc composée de trois modules (M1, M2, et M3). Une version simplifiée de M3 était donnée. Le reste de l'application a été découpé en 9 tâches, réparties dans les modules M1 et M2. Les communications entre les tâches respectent les mécanismes de communication Unix (tube/pipe, signaux, mémoire partagée). Il a été décidé de réaliser M1 et M2 à l'aide de processeurs ARM7 et M3 est un composant spécialisé (IP).

Dans ce cas d'étude, nous utilisons une partie du modem VDSL comme l'application. Cette application contient deux processeurs ARM 7 et un composant spécialisé. Le premier processeur exécute trois taches et le deuxième processeur exécute cinq tâches. Les communications entre les tâches et entre les processeurs et l'IP sont très intenses. La Figure 28 (a) montre la partie de modem VDSL implémentée et La Figure 28 (b) montre l'architecture de départ.



(a) La partie de modem VDSL implémenté **(b) L'architecture de départ**

Figure 28 Modem VDSL

4.3.3 Architecture logicielle matérielle

L'outil ROSES de group SLS TIMA permet de générer les interfaces logicielles et matérielles de cette application et de raffiner la spécification (architecture virtuelle) en une architecture RTL. Cet outil sera décrit en détaille dans le prochain chapitre (paragraphe 5.3.1). Au niveau RTL, on a deux processeurs ARM 7 et un IP. Chaque processeur ARM 7 exécute un logiciel se composé en logiciels d'applications (les tâches), un système d'exploitation minimal, et une couche d'abstraction du matériel. Les systèmes d'exploitation et les couches d'abstraction du matériel sont spécifiques aux processeurs et aux communications utilisées, mais aussi aux services requis par les tâches. Ils sont donc différents. Toutes les communications sont réalisées en utilisant des connexions point à point. Les protocoles utilisés sont variés : registre gardé ("guarded register"), poigné du main ("handshake"), et FIFO. La Figure 29 (a) décrit l'architecture logiciel/matériel de ce modem VDSL au niveau RTL. La Figure 29 (b) montre une autre représentation de cette architecture.

4.3.4 Objectif du prototypage et contraintes

Le flot de prototypage commence à partir de cette architecture. Nous utilisons la plateforme ARM Intergrator pour réaliser le prototype. Dans ce cas d'étude, l'objectif du prototypage est de vérifier les interfaces logiciel/matériel générées automatiquement par ROSES. On doit alors réaliser toutes les interfaces sur la plateforme.

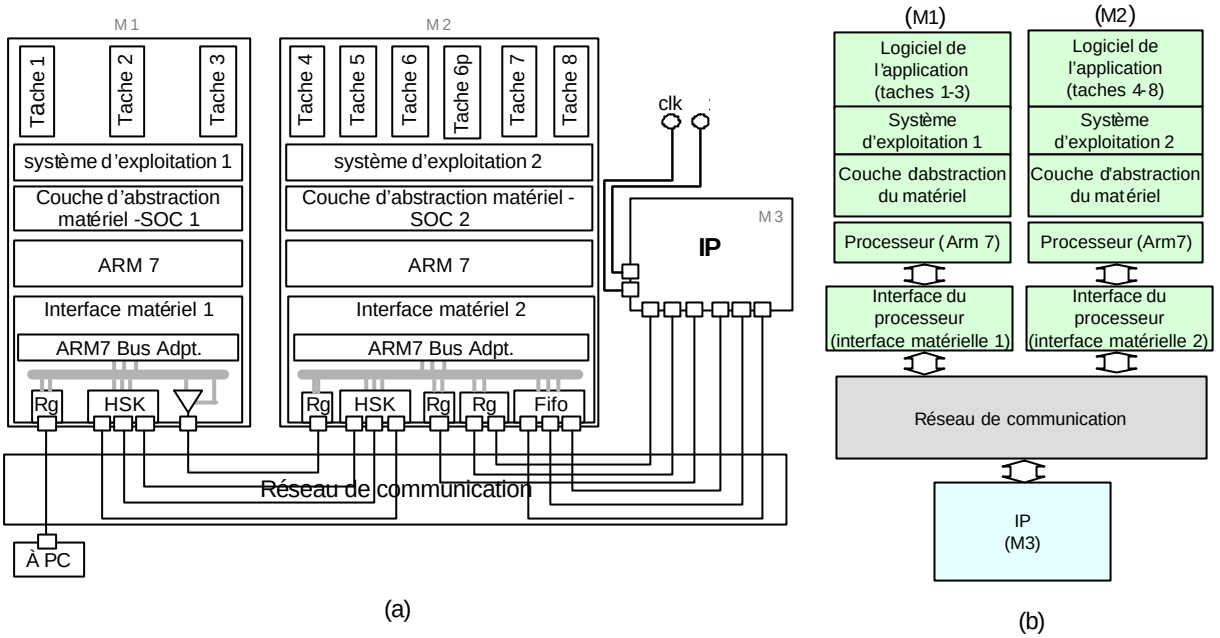


Figure 29 Architecture logiciel/matériel de modem VDSL au niveau RTL

La contrainte principale provient du réseau de communication de la plateforme qui est fixé. Le problème est de savoir comment réaliser les interfaces matérielles et sur quel nœud de prototypage.

4.3.5 Allocation

Dans l'étape d'allocation, on détermine les parties de la plateforme qui vont réaliser chaque partie de l'application. Puisque l'architecture de l'application se compose de deux processeurs ARM 7 et un IP, on décide alors d'utiliser deux modules de processeur avec ARM 7 pour réaliser ces deux processeurs. Pour l'IP, nous décidons de le réaliser sur le FPGA.

Pour la validation des interfaces, il faut les réaliser, en contournant le bus AMBA imposé par la plateforme. Nous avons alors décidé d'implémenter les interfaces matérielles et le réseau de communication sur le FPGA. Dans ce cas, les interfaces matérielles ne seront plus directement connectées aux processeurs, mais doivent d'être connectée au bus AMBA. La plateforme nécessite alors une phase de configuration pour cacher le bus AMBA. Concrètement, on introduit un convertisseur de protocole entre l'interface matérielle et le bus AMBA

L'allocation des parties logicielles est plus facile dans ce cas. Tout le logiciel du processeur M1 est exécuté sur le module du processeur 1 (nœud de prototypage logiciel 1), et tout le logiciel de processeur M2 est exécuté sur module processeur 2. Comme la plateforme ARM Integrator possède un plan d'adressage fixé différent de celui de l'application, on modifie l'application pour respecter cette contrainte imposée par la plateforme. Nous adaptons alors la couche d'abstraction du matériel. Le tableau 2 résume l'allocation de cette application.

L'application	La plateforme
Processeur M1	Carte du Processeur 1 (Arm7)
Processeur M2	Carte du Processeur 2 (Arm7)
Logiciel de M1	Exécuter par processeur 1
Logiciel de M2	Exécuter par processeur 2
IP (M3)	Module de FPGA
Les interfaces matériel	Module de FPGA
Réseau de communication	Module de FPGA
Configuration :	Convertisseur de bus AMBA
Adaptation :	L'adressage sur la couche d'abstraction matériel

Tableau 3 Allocation de l'architecture du VDSL sur la plateforme ARM Integrator

4.3.6 Configuration

Dans ce cas, la configuration consiste à ajouter un convertisseur de protocole pour cacher le bus AMBA de la plateforme. Une donnée du processeur vers l'interface matérielle traversera consécutivement le bus AMBA, puis le convertisseur (AMBA vers ARM7). La Figure 30 illustre cette configuration.

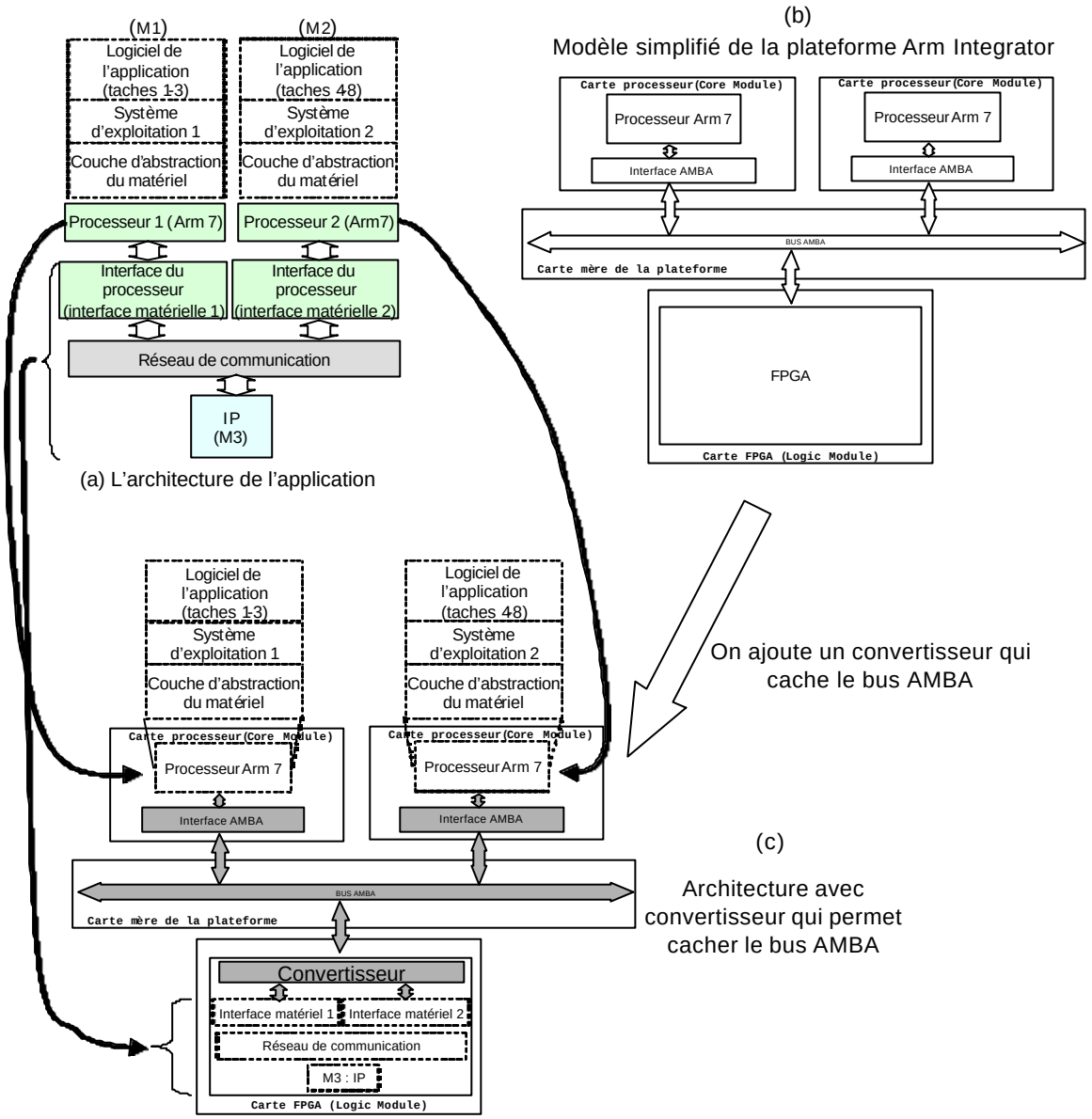


Figure 30 Configuration de l'application VDSL

La Figure 31 donne l'architecture de ce convertisseur. Le bus AMBA AHB est connecté à deux modules : un esclave par défaut et un pont AHB/APB à travers un décodeur adresses (pour le bus d'adresses), et un multiplexeur (pour le bus de données). Les interfaces matérielles sont connectées aux ponts AHB/APB à travers un bus APB. En plus de ces modules, on utilise aussi un contrôleur d'interruptions pour gérer et multiplexer des interruptions des interfaces matérielles car on n'a qu'une seule broche d'interruption à la sortie du FPGA.

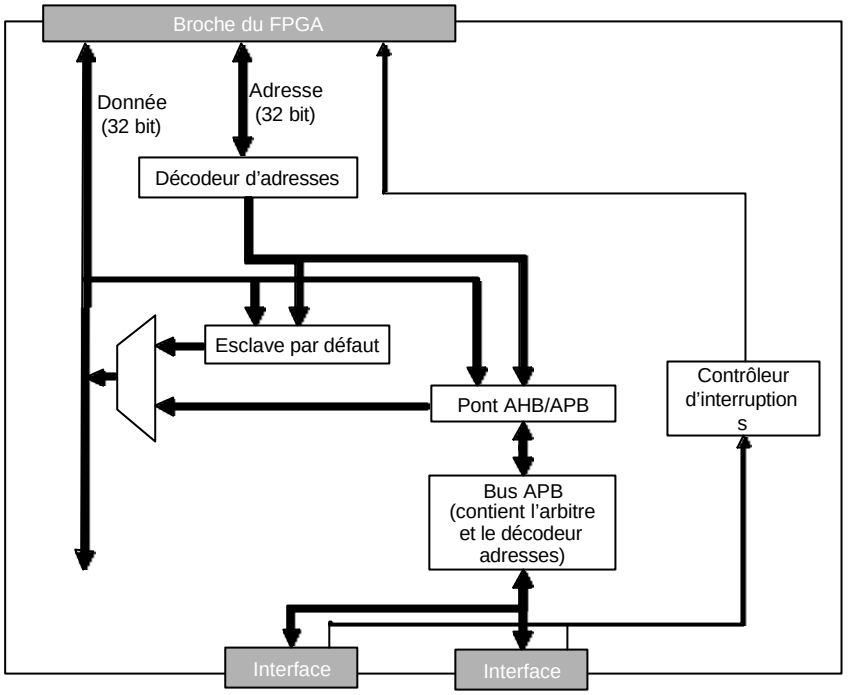


Figure 31 Convertisseur

Ce convertisseur a été développé à partir des spécifications du bus AMBA et des caractéristiques d'E/S des processeurs ARM 7, mais il est facilement réutilisable pour d'autres prototypes sur cette plateforme. Il faut ajouter ou supprimer des interfaces (en fonction du nombre d'interfaces matérielles) et modifier le décodeur d'adresses et l'arbitre de bus APB (un seul fichier VHDL synthétisable).

4.3.7 Adaptation

L'adaptation dans cette expérimentation est très simple et se décompose en deux parties. Il s'agit d'une part de modifier l'adresse de l'interface matérielle (des processeurs) pour respecter celle imposée par la plateforme pour tous les FPGA. D'autre part, un contrôleur d'interruptions qui multiplexe plusieurs sources d'interruptions est ajouté. Toutes ces modifications sont faites dans la partie logicielle d'abstraction du matériel.

4.3.8 Génération de code

On divise l'étape de génération de code en deux parties : la partie matérielle et la partie logicielle. La partie matérielle consiste en toutes les interfaces, le réseau de communication et l'IP que l'on met sur le FPGA. La partie logicielle consiste en programmes que l'on charge en mémoire pour les processeurs ARM 7.

Pour la partie matérielle, la génération de code consiste en une addition du module de plus haut niveau, suivie par la synthèse logique, et le placement routage. Le module de plus haut niveau est un fichier VHDL qui encapsule tous les composants et modules que l'on met sur le FPGA. La synthèse logique est effectuée en utilisant l'outil «FPGA compiler» de Synopsys. Enfin, le placement routage est effectué en utilisant l'outil « Quartus 2 » de Altera.

L'étape de génération de code pour la partie logicielle consiste en une addition d'un module de déboguage, la compilation et l'édition de lien. Le module de déboguage permet de connecter les programmes à un ordinateur pour afficher quelques messages de déboguage. La compilation et l'édition de lien sont effectués en utilisant les outils de développement de ARM (armcc, armcpp, armasm, et armlink). La Figure 32 montre le processus de génération de code.

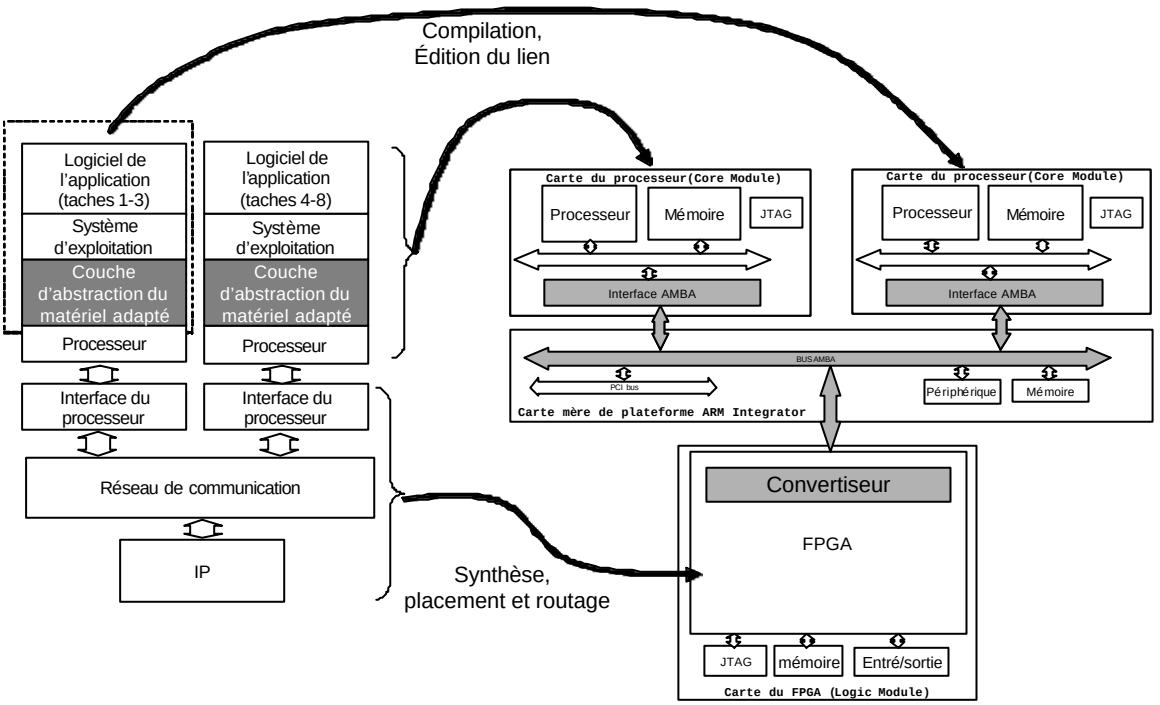


Figure 32 La génération de code

4.3.9 Résultats

La partie logicielle contient 106 fichiers (4915 lignes de code C/C++, et 597 lignes assembleur), et après l'étape de génération du code, elle occupe 24,9 K Octets de ROM et 1,8 K Octets de RAM. La partie matérielle utilise 7107 éléments logiques, et prend 18 % de

ressources du FPGA. La vitesse d'exécution de ce prototype est de 20 MHz (le processeur, la partie matérielle dans le FPGA, et le bus AMBA).

La vitesse d'exécution (20 MHz) est très supérieure à celle obtenue avec co-simulation en utilisant un ISS (quelques KHz). On décèle ainsi certains bogues qui apparaissent après de nombreux vecteurs de test, ce qui est trop long en co-simulation. Mais bien sur, l'exécution de ce prototype n'est pas précise au cycle prés.

La difficulté de cette expérimentation est la conception du convertisseur pour la configuration. En effet il faut connaître en détail le fonctionnement du bus AMBA et les protocoles d'entrées/sorties des processeurs. Néanmoins, ce convertisseur est largement réutilisable.

4.4 Prototypage du DivX

4.4.1 Description de l'application

Le DivX est une technologie pour compresser un fichier vidéo sans réduire la qualité visuelle. Cette technologie est basée sur le standard de compression MPEG 4. Elle permet de compresser un fichier au format MPEG-2 (le format utilisé sur le DVD) jusqu'à 10 % de sa taille d'origine. Malheureusement, l'encodeur est très complexe et l'implémentation temps réel devient extrêmement difficile. La Figure 33 montre le schéma de l'encodeur DivX [DIV01].

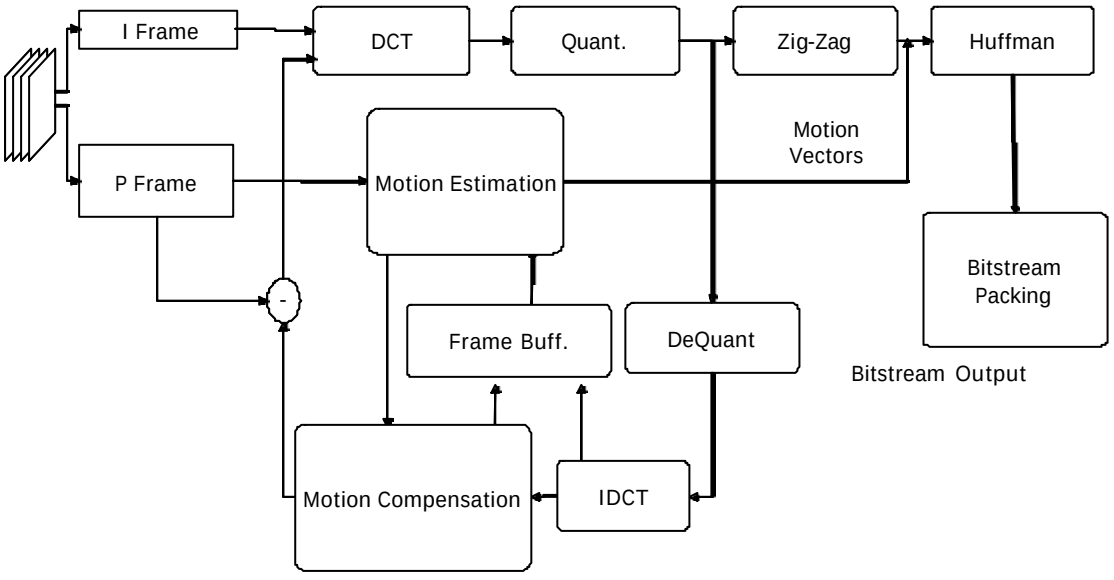


Figure 33 Schème de DivX encodeur

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

Le DivX est une application très complexe. La version gratuite de cette application est appelée OpenDivX. L'OpenDivX peut être compilé en utilisant un compilateur standard de C (comme gcc) et exécuté sur PC. Cette application contient environ 10000 lignes de code C. Le temps d'exécution de ce programme prend environ une seconde pour une séquence de 20 petites images (format QCIF : 176 x 144) sur un processeur Athlon 4 à 1200 MHz. C'est très loin des besoins de la plupart des formats vidéo qui nécessitent 30 images par seconde avec des formats beaucoup plus grands (CCIR 720 x 576, CIF 352 x 288, HDTV 1280 x 720 – 60 images par seconde).

4.4.2 Objectif du prototypage et contraintes

Nous voulons développer un système multiprocesseur monopuce de l'OpenDivX pour l'encodage en temps réel. Afin d'accélérer le temps d'exécution, le code de l'application doit être parallélisé et son exécution doit être partagée entre plusieurs processeurs. Une mécanisme de communication doit être alors inséré.

Avant de réaliser cette application sur une puce, on souhaite développer un prototype sur une plateforme ARM Integrator. Les objectifs sont (1) de vérifier le code parallélisé, (2) de vérifier le système d'exploitation pour être sur qu'il marche sur une configuration multiprocesseur, et (3) d'insérer un mécanisme de communication.

Le prototype n'aura pas la même performance que le système monopuce final. En effet, la fréquence d'horloge du processeur sur la plateforme ARM Integrator est beaucoup plus lente (20 MHz), et le réseau de communication sur la plateforme est fixé à un bus AMBA qui a une bande passante très limitée.

4.4.3 La spécification et l'architecture RTL

L'encodeur divX est partitionné en deux modules avec une configuration maître et esclave (Figure 34). Le maître effectue le processus du calcul de plus haut niveau. L'esclave effectue le calcul du vecteur de mouvement et la compensation du mouvement. Un protocole MPI [MPI95] (Anglais : Message Passing Interface) est choisi comme protocole de communication.

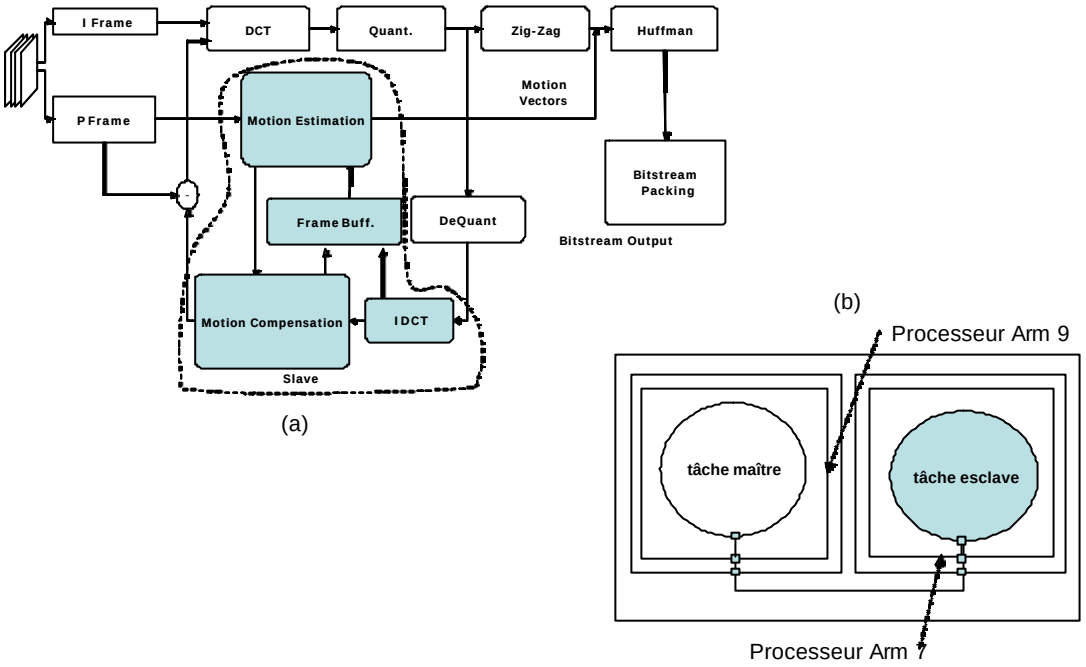


Figure 34 La spécification de départ de l'application DivX

MPI est une interface standard pour la communication entre plusieurs tâches sur une machine parallèle. Dans notre application, nous n'implémentons que deux primitives : *MPI_send* et *MPI_receive*. Ces deux fonctions permettent de réaliser des communications bloquantes entre les taches en utilisant des FIFO. Chaque paire d'émetteur/récepteur a besoin de deux FIFO : une FIFO pour garder le message, et la deuxième FIFO pour garder le *tag*. Le *tag* est un identificateur du message.

La Figure 35 montre l'architecture logiciel/matériel de l'application. La partie logicielle correspond aux tâches, aux systèmes d'exploitation, aux parties logicielles de l'implémentation du protocole MPI, et aux couches d'abstraction du matériel. La partie matérielle consiste en deux processeurs et au réseau de communication comprenant la partie matérielle d'implémentation du protocole MPI.

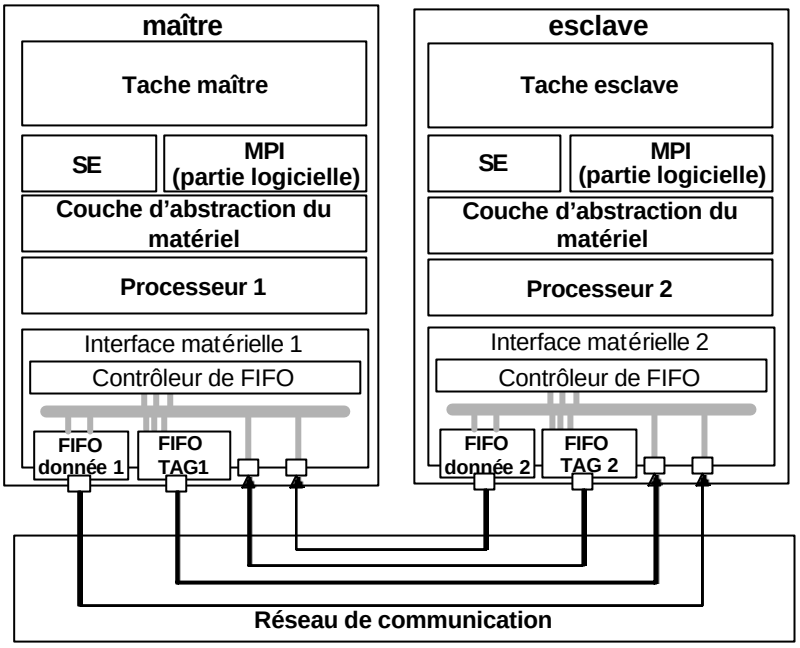


Figure 35 L'architecture logicielle/matériel de l'application DivX

Dans le standard MPI, les tâches (logiciel de l'application) émettent ou reçoivent des données. Un message est un ensemble de données auquel s'ajoute des informations de contrôle (entête, CRC,…). Un message est découpé en cellules. La taille d'une cellule est fixée par la taille du bus de données. Pour profiter d'un mode de communication par paquets (mode burst). On regroupe les cellules en paquets pour profiter des performances du réseau. La Figure 36 illustre ce concept.

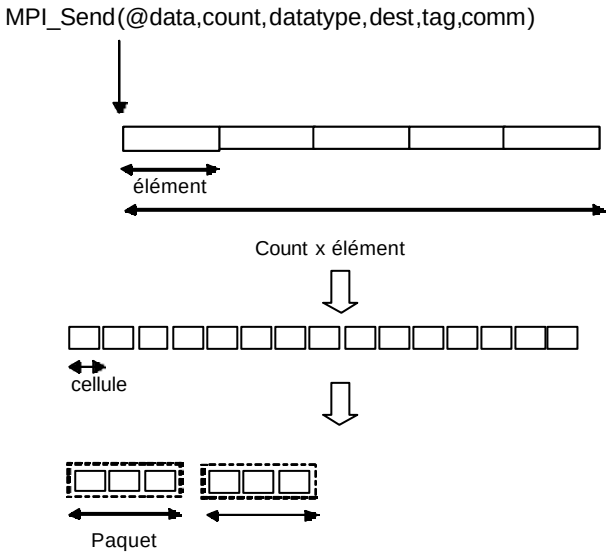
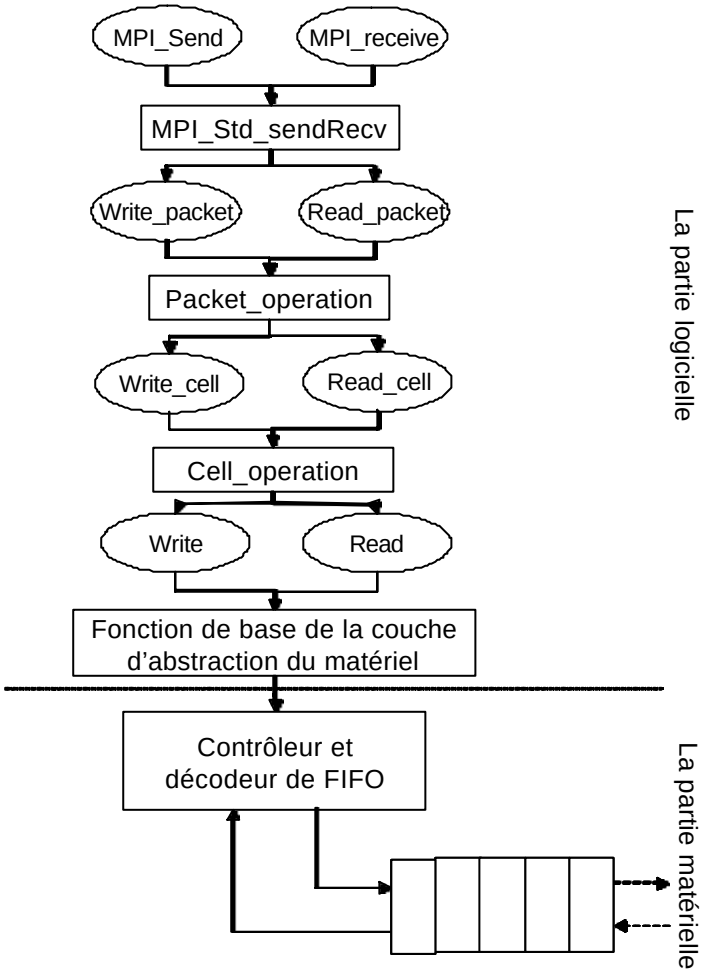


Figure 36 Décomposition des donnée

La Figure 37 montre notre implémentation de la communication. Cette implémentation est le résultat de la génération automatique en utilisant ASAG et ASOG. Sur la partie logicielle, les fonctions *MPI_Send* et *MPI_Receive* sont fournies par l'élément *MPI_Std_SendRecv* de la bibliothèque d'ASOG. Cet élément utilise des fonctions *Write_packet* et *Read_Packet* qui sont fournis par l'élément *packet_operation*. L'élément *packet_operation* a besoin des fonctions *read_cell* et *write_cell* fournies par l'élément *cell_operation*. Enfin, l'élément *cell_operation* utilise des fonctions *read* et *write* (de *la couche d'abstraction du matériel*) pour envoyer les données à la FIFO à travers le contrôleur et le décodeur d'adresses de la FIFO. Le contrôleur de la FIFO gère l'état de la FIFO (la tête, la queue) et informe le processeur si il y a un changement d'état de la FIFO (par exemple si la FIFO devient pleine ou vide).



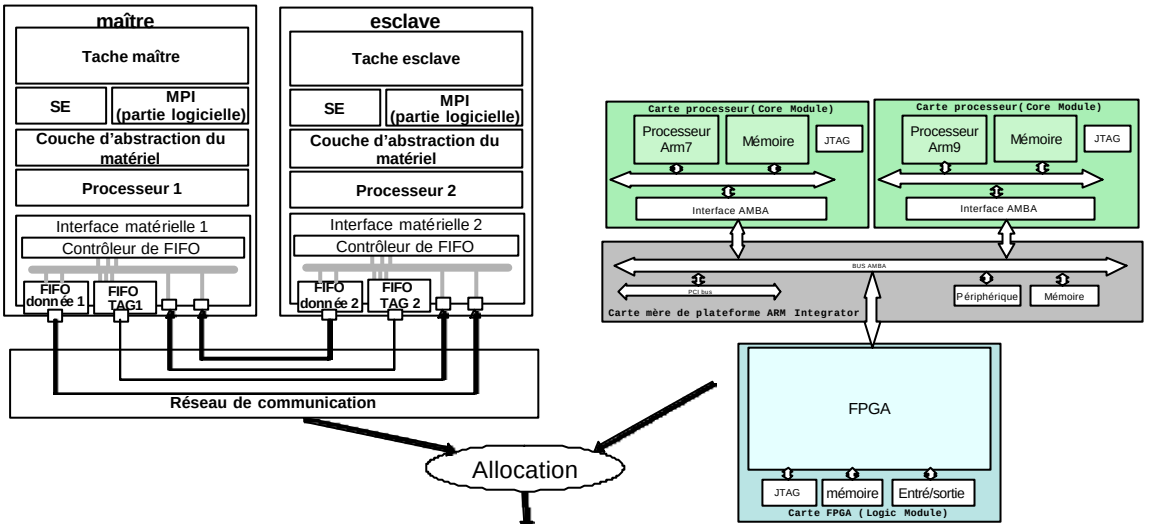
(b) Notre implémentation de communication

Figure 37 Implémentation de la partie communication de l'exemple DivX

4.4.4 Allocation

A l'étape d'allocation, nous décidons de réaliser le plus possible de fonctions en logiciel. Cette décision est dirigée par les contraintes de la plateforme ARM Integrator où le réseau de communication est fixé. Toutes les fonctionnalités de MPI sont réalisées en logiciel parce que l'on ne peut pas ajouter du matériel connecté directement aux processeurs.

La Figure 38 illustre le processus d'allocation. Il s'agit de faire correspondre chaque partie de l'application avec une partie de la plateforme. Dans cette application, les deux processeurs (processeur 1 pour le maître, et processeur 2 pour l'esclave) sont alloués à des cartes processeurs (avec ARM 9 pour le maître et ARM 7 pour l'esclave). Les logiciels sur ces processeurs sont respectivement exécutés par ces processeurs.



Composant/Element	Allocation
Processeur 1	ARM 9
Processeur 2	ARM 7
Tache maître	Exécuté par ARM 9
Tache esclave	Exécuté par ARM 7
Contrôleur de FIFO	Distribué dans Proc 1 & 2
FIFO	Mémoire partagée
Non utilisé	Logic module

Tableau d'allocation

Figure 38 Allocation de DivX

Le problème que l'on doit résoudre, provient de la partie de communication. On ne peut pas réaliser les FIFO et ses contrôleurs en matériel à coté des processeurs. Pour obtenir le prototype rapidement, les FIFO sont réalisées dans la mémoire partagée, et les fonctionnalités de son contrôleur sont émulées par logiciel.

4.4.5 Configuration

Il n'y a pas de configuration dans cette expérimentation. Toutes les difficultés sont résolues par l'adaptation.

4.4.6 Adaptation

Après l'étape d'allocation, il existe encore une différence entre l'architecture du système et celle de la plateforme ARM Integrator pour la partie communication. Le système utilise des connexions point-à-point avec FIFO encapsulées par le protocole MPI. La plateforme supporte uniquement un bus AMBA AHB pour la communication entre les nœuds de prototypage.

Afin de résoudre ce problème, nous réalisons le comportement de la partie communication en logiciel en gardant le même protocole. C'est-à-dire que les FIFO sont réalisées sur les mémoires partagées, et les contrôleurs sont distribués dans les couches d'abstraction du matériel. Comme nous souhaitons valider l'application et le système d'exploitation, une réalisation différente de la communication ne devrait pas poser de problème du point de vue de l'application.

Il s'agit donc de modifier les couches logicielles de bas niveau (couche d'abstraction du matériel). Cette modification semble assez difficile car d'une part une partie de cette couche est écrite en assembleur et nécessite de bien connaître le caractéristiques des cartes de prototypage (processeur et FPGA), et d'autre part, cette couche logicielle est issue d'un processus de génération automatique à l'aide des outils développés dans le groupe SLS. Pour cette raison, j'ai modifié les outils de génération automatique, ce qui facilite cette étape d'adaptation. Ceci est détaille au chapitre 5.

Le processus de cette adaptation consiste à revenir à la spécification de plus haut niveau, à modifier cette spécification, et à régénérer une nouvelle couche d'abstraction du matériel. La Figure 39 montre cette spécification. En effet, ce que nous faisons est de revenir à l'abstraction précédente, car cette architecture est le résultat du raffinement de cette spécification.

Dans cette spécification, chaque processeur possède une tâche : tâche maître et tâche esclave. Ces tâches demandent des services de communication : *MPI_send* et *MPI_receive*. Ces services sont indiqués dans les paramètres de port des tâches (service = *MPI_send*).

L'implémentation de la partie logicielle de ce service est faite en utilisant l'élément *packet_operation* de la bibliothèque (Figure 37). La partie matérielle utilise des FIFO. Ces éléments sont décrits sur les paramètres des ports externe et interne des processeurs.

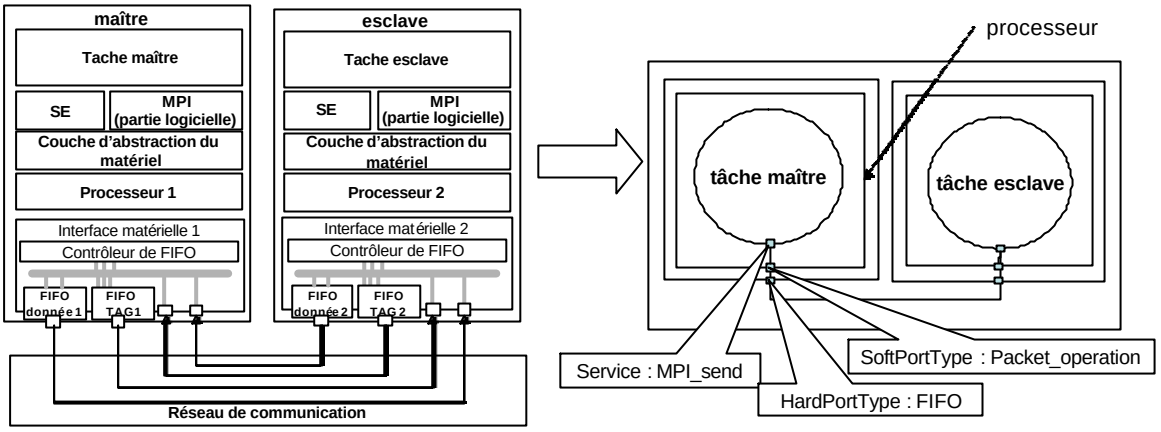


Figure 39 Revenir à la spécification plus haut niveau

Afin de générer l'interface logicielle adaptée à la plateforme, on doit modifier les paramètres dans cette spécification. Le paramètre de *SoftPortType* est remplacé par *Packet_operation_on_plateforme.* Ce paramètre indique l'implémentation de la communication du coté logiciel. Coté matériel, nous ne générons pas d'interface matérielle, nous utilisons le matériel existant de la plateforme.

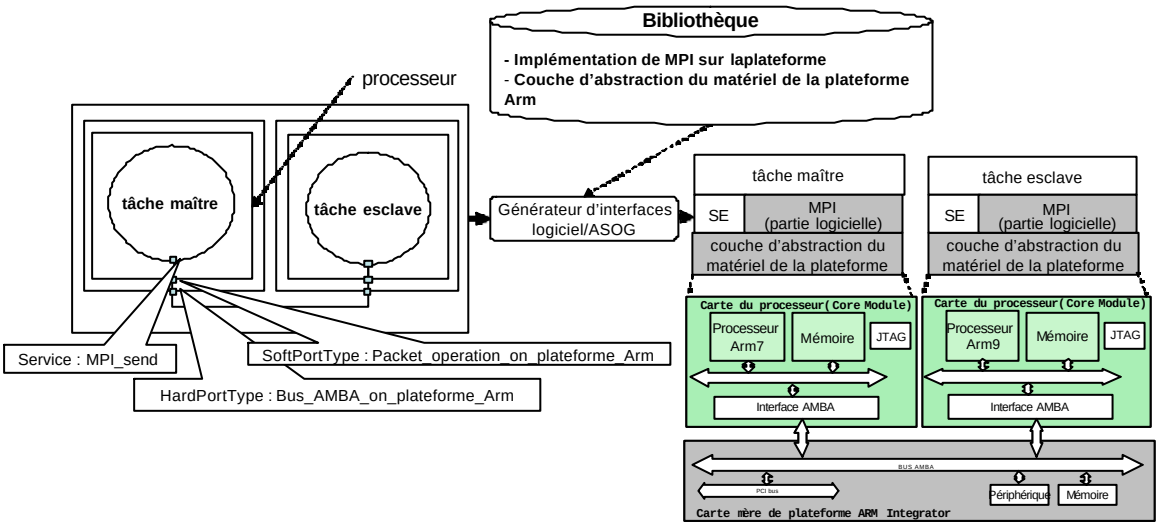


Figure 40 Génération d'interface logicielle

La Figure 41 illustre la réalisation de la communication sur la plateforme. Du point de vue du logiciel de l'application, la communication a le même comportement. Mais la

réalisation de cette fonction est différente. Les fonctions *write_cell* et *read_cell* envoyent la donnée à la FIFO sur la mémoire à la place de la FIFO physique. Ces fonctions logicielles doivent gérer cette FIFO, c'est-à-dire mettre à jour les registres qui gèrent le nombre d'éléments stocké dans à FIFO, l'adresse de la tête, et l'adresse de la queue de la FIFO. En plus, ces fonctions envoyent une interruption à l'autre processeur quand ils changent l'état de la FIFO : à partir de l'état plein à l'état non plein (quand on récupère des données/*read_packet*), et à partir de l'état vide à l'état non vide (quand on envoie des données/*write_packet*). On a besoin alors de fonctions permettant d'envoyer des interruptions aux autres processeurs. Cette fonction est une partie de la couche d'abstraction du matériel.

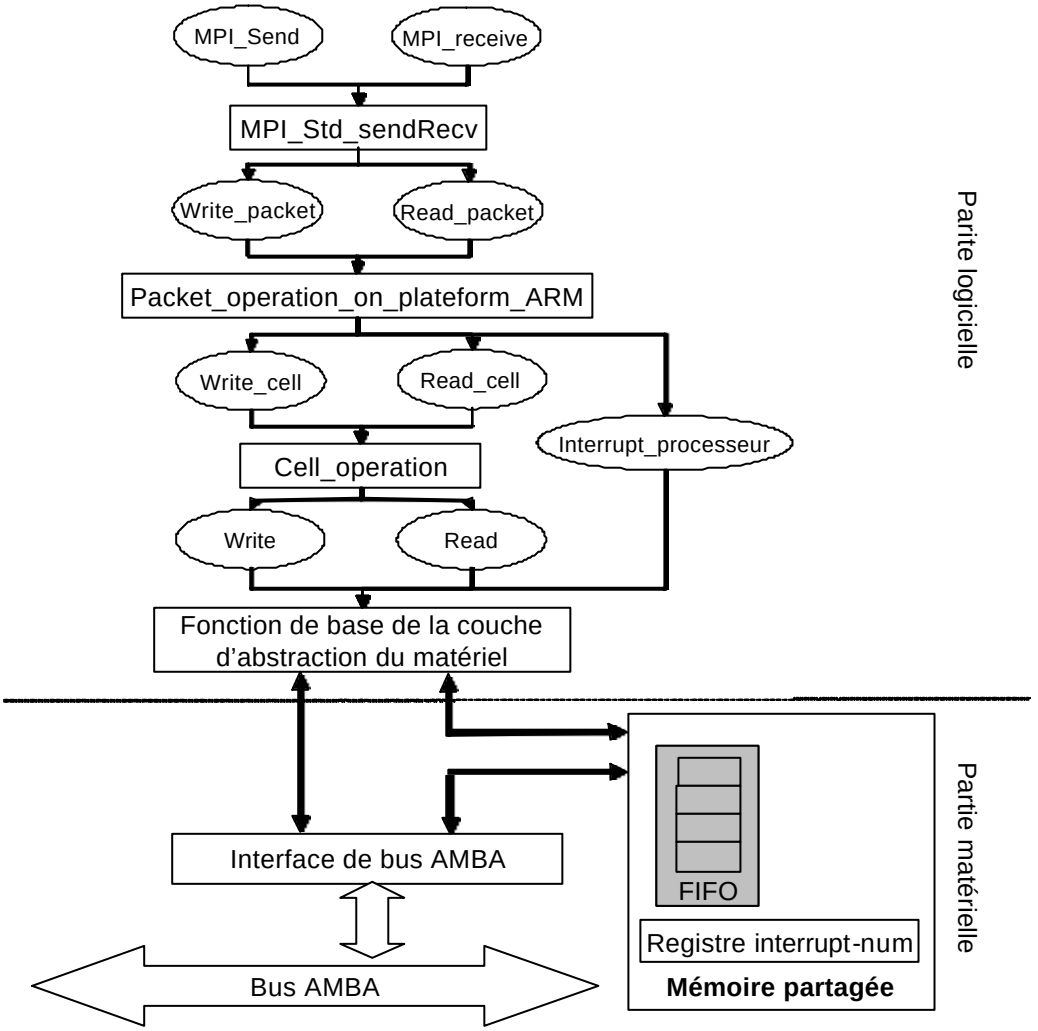


Figure 41 Implémentation de la partie communication sur le prototype de l'application DivX

4.4.6.1 L'adaptation de la couche d'abstraction du matériel pour la partie MPI

Comme ce qui est illustré dans la Figure 41, le protocole MPI est réalisé en matériel et en logiciel dans le système monopuce finale. La partie logicielle effectue deux tâches : couper (ou fusionner) les messages envoyés à la taille de la FIFO et (2) envoyer ces messages à la FIFO. La partie matérielle consiste en un décodeur d'adresses, deux FIFO et un contrôleur de FIFO. Le décodeur d'adresses sélectionne les FIFO, le contrôleur de FIFO enregistre et met à jour l'état de la FIFO (la tête, la queue, et le nombre d'éléments mémorisé dans la FIFO). Ce contrôleur envoie des interruptions aux processeurs s'il y a des changements d'état de la FIFO : de l'état plein à l'état non plein ou de l'état vide à l'état non vide.

On ne peut pas connecter directement un FPGA/matériel aux processeurs (via le bus AMBA) sur la plateforme, aussi toutes les fonctions requises par MPI sont faites en logiciel ou plus précisément par une fonction dans la couche d'abstraction du matériel. La fonctionnalité de la partie logicielle de MPI n'est pas changée. Dans cette couche logicielle, les FIFO sont placées sur la mémoire partagé. On ne doit pas alors ajouter de FIFO matérielles et le décodeur d'adresses pour ces FIFO. Les fonctionnalités du contrôleur de FIFO sont distribuées en logiciel : La mise à jour des registres qui indique le nombre d'éléments dans la FIFO est faite par les émetteurs/récepteurs dans la même temps ou ils accèdent à la FIFO. L'interruption est envoyée de l'émetteur au récepteur si la FIFO n'est plus vide. Ça se fait si l'émetteur envoie des nouvelles données à une FIFO vide. L'interruption est envoyée du récepteur à l'émetteur si la FIFO n'est plus pleine. Ca se fait si le récepteur retire des données d'une FIFO pleine.

4.4.6.2 L'adaptation de la couche d'abstraction du matériel pour un autre matériel

Les différences entre la partie matérielle de l'application et de la plateforme entraînent quelques adaptations aussi sur la couche d'abstraction du matériel. Les différences principales sont : le contrôleur d'interruption, et le plan de mémoire.

Dans l'application finale sur puce, le processeur ne doit pas envoyer des interruptions aux autres processeurs car le contrôleur de FIFO gère l'état de la FIFO. Dans le prototype, chaque processeur a besoin d'envoyer des interruptions aux autres processeurs qui communiquent avec lui s'il change l'état de la FIFO. Le processeur a alors besoin d'un espace dans la mémoire partagée pour communiquer le numéro d'interruption entre deux

processeurs. Ces besoins entraînent : une fonction pour interrompre un autre processeur, une sous-routine de traitement d'interruption pour contrôler le numéro d'interruption à la place du registre dans le contrôleur de FIFO, et une synchronisation entre processeurs.

4.4.7 Génération du code

Le processus de raffinement génère les codes pour la couche d'abstraction du matériel, le système d'exploitation et MPI. Ces codes et le code de tâches sont compilés, et téléchargés sur le carte de processeur. Les codes C sont compilés en utilisant armcc, les codes C++ avec armcpp, les codes assembleur avec armasm. Enfin, nous faisons de l'édition du lien de tous les codes pour avoir une image exécutable pour chaque processeur.

4.4.8 Résultats et analyse

L'expérimentation montre que le prototype peut être développé en moins de deux semaines si la plupart des bibliothèques sont prêtes. Ces bibliothèques sont réutilisables, par conséquent, nous ne devons les développer qu'une seule fois. Le prototype nous a donné le moyen d'atteindre notre objectif : développer le logiciel de l'application (les tâches). Nous avons validé le code parallélisé de l'application. De nombreux de bogues ont été trouvés et corrigés. Le prototype traite environ 100 images au format QCIF par heure (3 images par l'heure si on utilise une ISS).

Le système d'exploitation a été partiellement validé pour tourner sur la configuration multiprocesseur. La plupart des problèmes que nous avons corrigés sont liés à la synchronisation et au partage de ressources entre les processeurs. Il n'est pas facile trouver ce genre de problèmes avec la simulation parce qu'ils ne se produisent pas à chaque exécution. Par conséquent, cela prend un long temps d'exécution pour détecter ce problème en utilisant la simulation.

CHAPITRE 5 : ANALYSE ET PERSPECTIVES

5.1 Introduction 84

5.2 Evaluation des exemples 84

5.2.1 VDSL 84

5.2.2 DivX 85

5.2.3 Synthèse 86

5.3 Automatisation de l'adaptation pour accélérer le développement du logiciel 87

5.3.1 Les outils de génération automatique du flot ROSES 87

5.3.2 L'automatisation de l'adaptation 93

5.3.3 Avantages et limitations 96

5.3.4 Conclusion sur l'adaptation 98

5.4 Co-émulation 98

5.4.1 Exemple de co-émulation en utilisant la plateforme ARM Integrator 98

5.4.2 Evaluation 104

5.4.3 Perspectives 105

5.1 Introduction

Le chapitre 4 a décrit deux applications prototypées sur la plateforme ARM Integrator en utilisant le flot de prototypage donné dans le chapitre 3.

Ce chapitre cherche maintenant à évaluer ces exemples, et voir dans quels cas il est possible de réaliser un prototypage automatique. Pour cela, on s'intéresse plus particulièrement à l'étape adaptation dont l'automatisation n'est pas triviale. On termine ce chapitre par une extension de la proposition à la co-émulation.

5.2 Evaluation des exemples

5.2.1 VDSL

Le premier exemple donné est l'application VDSL. Dans ce cas d'étude, nous voulions vérifier les fonctionnalités des interfaces en utilisant le prototypage. Afin d'obtenir le prototype, nous avons effectué les quatre étapes du flot de prototypage.

La première étape est l'allocation. Dans cette étape, la principale difficulté est d'avoir un réseau de communication différent entre la plateforme et l'application. Pour vérifier la fonctionnalité des interfaces, il faut introduire un modèle du réseau de l'application dans le nœud de prototypage matériel. Ce qui nécessite aussi un convertisseur entre le protocole du réseau de communication imposé par la plateforme et celui de l'application. Il faut donc modifier la couche logicielle d'abstraction du matériel. C'est dans cette étape d'allocation que sont prises les décisions concernant l'adaptation et la configuration. La principale difficulté présentée par cette application peut être résolue en introduisant un convertisseur de protocole ; ceci concerne la configuration de la plateforme.

La deuxième difficulté est la partie matérielle différente entre la plateforme et l'application. Ces différences sont concernant le contrôleur d'interruption et le plan mémoire. Ce problème se résout en modifiant le modèle RTL, notamment la couche d'abstraction du matériel de l'application.

L'étape suivante est la configuration. Pour l'application VDSL, on a développé un convertisseur pour cacher le bus AMBA et l'interface AMBA. Le modèle de ce convertisseur est synthétisable, puisqu'il est réalisé dans le FPGA d'un nœud de prototypage matériel. Il a été écrit d'une façon générique et peut se connecter à plusieurs processeurs

ARM de type différent (ARM 7 et ARM 9). La conception de ce convertisseur n'est pas difficile et il est réutilisable sur une même plateforme.

L'adaptation consiste à modifier une partie de l'application qui est généralement la couche logicielle de plus bas niveau. En effet, il n'est pas possible d'insérer une autre couche logicielle pour détourner les incompatibilités, puisque cette couche logicielle communique directement avec le matériel. Dans le cas du VDSL, cette modification n'est pas très difficile, mais le concepteur doit connaître toutes les adresses utilisées dans le code (assembleur généralement). Il peut aussi être amené à modifier les routines de traitement des interruptions. Aussi, une automatisation faciliterait cette phase de prototypage.

La dernière étape est la génération de code pour laquelle de nombreuses phases sont déjà automatisées : synthèse, placement routage, compilation, édition de lien, chargement. Dans la partie matérielle, il faut cependant ajouter le module de plus haut niveau, et dans la partie logicielle, il faut ajouter des fonctions de déboguage. Ces deux tâches ne sont pas automatisées, mais elles ne présentent que peu de difficulté.

5.2.2 DivX

Dans la deuxième expérience, nous avons développé le prototype de l'application DivX pour faciliter le développement du logiciel de l'application. Dans ce cas, nous ne nous intéressons pas au détail de réalisation de la partie matérielle, ni de la couche logicielle de bas niveau. Comme on souhaite valider les tâches, il faut respecter la spécification de l'échange des informations entre les tâches, indépendamment de la réalisation physique et des protocoles. On n'a donc pas besoin ici que le réseau de communication de la plateforme soit proche ou identique à celui de l'application finale.

Dans l'étape d'allocation, le problème porte toujours sur le réseau de communication de la plateforme différent de celui de l'application. Même si l'objectif du prototypage, la validation du logiciel de l'application n'impose pas d'avoir le même réseau de communication, il faut modifier l'application pour utiliser le bus AMBA de la plateforme et les mémoire partagées, et émuler les fonctionnalités du contrôleur de FIFO en logiciel. Tout ceci entraîne d'importantes modifications dans la partie logicielle d'abstraction du matériel. Les décisions prises pendant l'allocation guide le reste du processus de prototypage.

Dans cet exemple, il n'y a pas de configuration car on n'a pas besoin de réaliser le réseau de communication étant donné l'objectif du prototypage.

La difficulté dans le prototypage du DivX était la modification des couches logicielles de bas niveau dépendant du matériel pour réaliser les protocoles de communication MPI. A cela s'ajoute le fait de ne pas utiliser le nœud de prototypage matériel, ce qui empêche de réaliser les FIFO de façon matérielle. Il a donc fallu trouver une nouvelle méthode pour réaliser les fonctions MPI entièrement en logiciel, et réécrire toute la partie logicielle de bas niveau. On a donc essayé de trouver une technique pour automatiser ces transformations délicates et source d'erreurs.

Comme ces couches logicielles de bas niveau sont obtenues automatiquement avec l'outil ASOG à partir d'une spécification du système, on a donc réorganisé la spécification et généré ces couches logicielles adaptées à la plateforme. Mais cette transformation n'est pas aisée. Cette technique est décrite dans le paragraphe 5.3.

Comme dans l'exemple précédent, l'étape de génération du code est entièrement automatisée.

5.2.3 Synthèse

Enfin, on peut conclure que :

- La décision principale est prise dans l'étape d'allocation. Cette décision dépend des contraintes imposées par la plateforme, et de l'objectif du prototypage.
- Dans l'étape de configuration et d'adaptation, on implémente les décisions prises dans l'étape d'allocation.
- Le convertisseur utilisé dans l'étape de configuration est réutilisable sur la même plateforme.
- L'adaptation est une étape très laborieuse. L'automatisation est nécessaire pour simplifier cette étape.
- L'étape de génération du code est déjà automatique.

5.3 Automatisation de l'adaptation pour accélérer le développement du logiciel

Le développement du logiciel est devenu un goulot d'étranglement dans le flot de conception des systèmes monopuces. En effet, il représente une part très importante du temps de développement du système, mais aussi, il ne peut être validé que si l'on a le matériel pour l'exécuter. Ce qui signifie que les concepteurs retardent sa validation jusqu'à l'obtention du circuit. Aussi, une plateforme de prototypage est une alternative si celle-ci est utilisée comme un environnement pour développer ce logiciel. Elle permet alors d'accélérer la conception du système.

Dans le chapitre précèdent, nous avons montre un exemple de prototypage de l'application DivX (paragraphe 4.4). Ce prototypage avait pour objectif de valider le logiciel de l'application le plus tôt possible. L'objectif de ce paragraphe est alors de présenter l'automatisation de l'adaptation en utilisant le flot ROSES.

Ce paragraphe décrit le flot ROSES, ainsi que son utilisation pour l'étape d'adaptation. On discute ensuite des avantages et des limitations de cette technique.

5.3.1 Les outils de génération automatique du flot ROSES

Le flot ROSES du groupe SLS permet la génération automatique d'interfaces logiciel/matériel des systèmes monopuces. L'interface logiciel/matériel (Figure 42) générée est un adaptateur de communication entre les taches exécutées par le processeur et les ressources matérielles. Plus précisément l'interface logicielle permet au code de l'application d'accéder aux ressources matérielles du processeur (typiquement pour faire des E/S) et l'interface matérielle permet au processeur d'accéder au réseau de communication.

Les interfaces matérielles prises en compte par ROSES concernent les processeurs [LYO03], mais aussi la mémoire [GHA03], et les composants matériels spécifiques [GRA04]. Les interfaces logicielles sont les systèmes d'exploitation [GAU01] et les couches de communications [PAV04]. L'intérêt de ROSES est la génération automatique des interfaces à partir d'une spécification du système et de quelques caractéristiques pour guider la conception (protocoles, adressage, et autres paramètres). De plus, ce flot permet une validation à plusieurs niveaux d'abstraction à l'aide de mécanismes de co-simulation [NIC02].

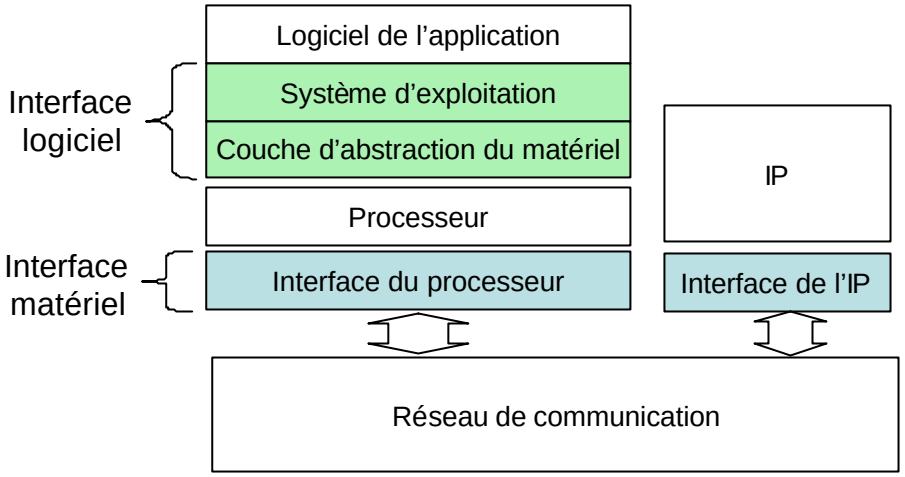


Figure 42 Interface logiciel/matériel

Le principe le plus important dans ce flot est la conception d'un système complet par assemblages d'éléments [CES02]. Que ce soit pour composer les parties logicielles des interfaces ou les parties matérielles mais aussi le développement des modèles de simulation, la technique reste la même : cela consiste en un assemblage d'éléments de bibliothèque.

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

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

La Figure 43 montre le flot qui permet la génération d'interfaces à partir d'un système décrit au niveau architecture vers une description au niveau RTL. L'entrée du flot est décrite en langage de spécification développé par le groupe SLS. Ce langage est une extension de SystemC [SYS00] nommé VADeL pour *Virtual Architecture Description Language* pour faire référence au concept de modules virtuels décrit prochainement.

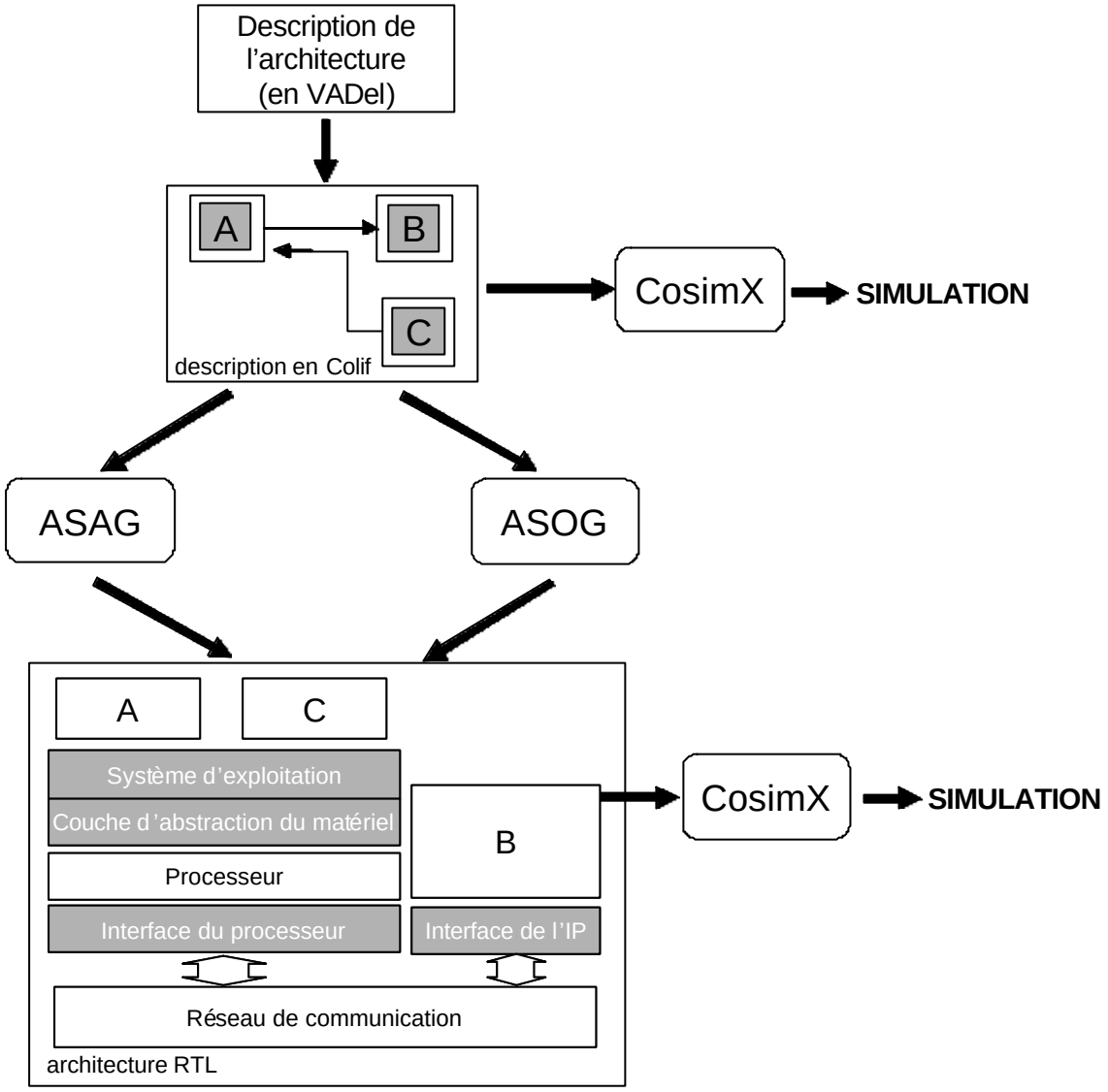


Figure 43 le flot ROSES

VADel [CES01] est un langage construit à partir de C++ : il est donc orienté objet. Le C++, avec son concept de classes d'objets permet de développer facilement de nouvelles structures de données. En fait, VADeL n'est pas une extension directe de C++, mais c'est une extension de SystemC, un langage de description de matériel pour la simulation.

La première étape du flot consiste à traduire la spécification en VADel annotée en COLIF [CES01]. La spécification en COLIF décrit un système comme un ensemble d'objets interconnectés de trois types : les modules, les ports et les liens (en anglais : net). Un objet, quel que soit son type, est composé de deux parties. La première partie, nommée «entity», permet la connexion avec les autres objets. La deuxième partie, nommée «content», fournit

une référence à un comportement ou des instances d'autres objets. Cette possibilité d'instancier des éléments permet le développement de modules, ports et liens hiérarchiques.

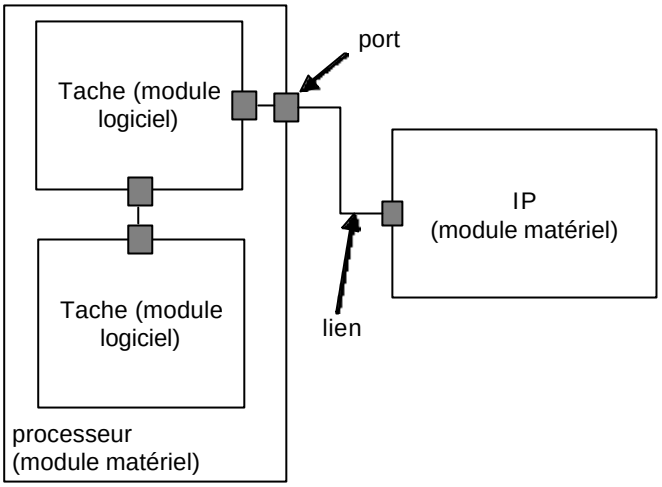


Figure 44 Un exemple de modules, de liens, et de ports

La Figure 44 montre un modèle décrit en COLIF. Les boites carrées représentent les modules, les petits boites foncées représentent les ports, et les lignes représentent les liens. Les fonctionnalités de chaque élément sont les suivantes :

1. Le module représente un composant matériel ou logiciel. Son «entity» donne son type et les ports par lequel il communique. Son «content» peut, être caché (boite noire), contenir des références à un comportement (ex : fichier C ou VHDL/matériel connu), mais peut aussi être un sous-système décrit sous forme de modules, ports et liens.
2. Le port représente un point de communication pour un module. Son «content» est composé de deux parties : l'une est en relation avec l'intérieur du module et l'autre avec l'extérieur. Ces deux parties peuvent être cachées, contenir des références à un comportement ou des instanciations de ports.
3. Le lien (Anglais : net) représente une connexion entre plusieurs ports. De la même façon, un lien peut contenir des références à un comportement ou contenir d'autres liens.

En utilisant cette description COLIF, le système est décrit au niveau architecture pour l'entrée du flot ROSES. En ce point, le système est décrit comme un ensemble de modules virtuels communicants par des canaux virtuels à travers des ports virtuels (Figure 45).

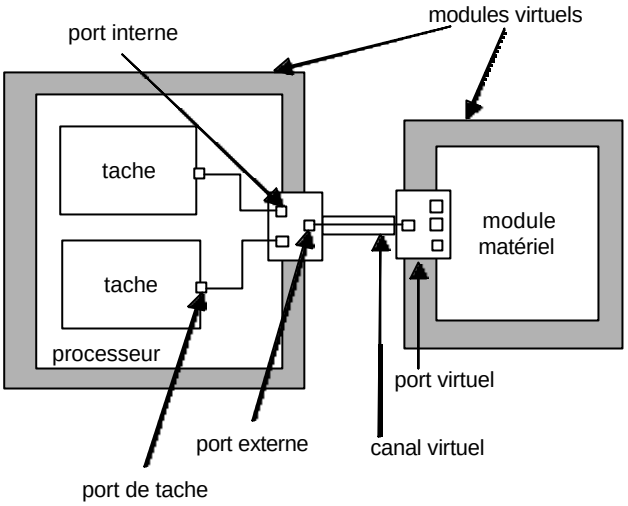


Figure 45 Un exemple d'entée du flot ROSES

Un canal virtuel est un canal de communication de haut niveau d'abstraction (on utilise aussi le terme de canal abstrait) qui représente un ou plusieurs canaux de bas niveau d'abstraction. Les caractéristiques de ce canal sont connues (le protocole utilisé, la taille du bus, la taille de la FIFO, ….) mais les détails de réalisation n'apparaissent pas explicitement (le bus de données, les signaux de contrôle, d'adresses, …). On utilise la notion de lien en COLIF pour spécifier le canal virtuel.

Ce canal virtuel permet la communication entre les modules virtuels. Le module virtuel encapsule des composants. Ce composant virtuel accède au canal virtuel en traversant un port virtuel. Un port virtuel est composé de ports internes (connectés aux composants) et de ports externes (connectés aux canaux virtuels). Les ports virtuels adaptent les protocoles et les niveaux d'abstractions entre les ports externes et les ports internes. Ceci permet aux composants de modules virtuels de ne pas être décrits au même niveau d'abstraction, et de pouvoir s'échanger des informations.

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

L'outil ASOG a été développé par Lovic Gauthier dans son travaille de thèse [GAU01]. Cet outil génère l'interface logicielle qui consiste en un système d'exploitation

minimal et la couche d'abstraction du matériel à partir de paramètres donnés dans les ports. Les paramètres des ports de taches décrivent les services de communication demandés et requis par la tache. Les paramètres de ports internes indiquent l'implémentation de l'interface logicielle.

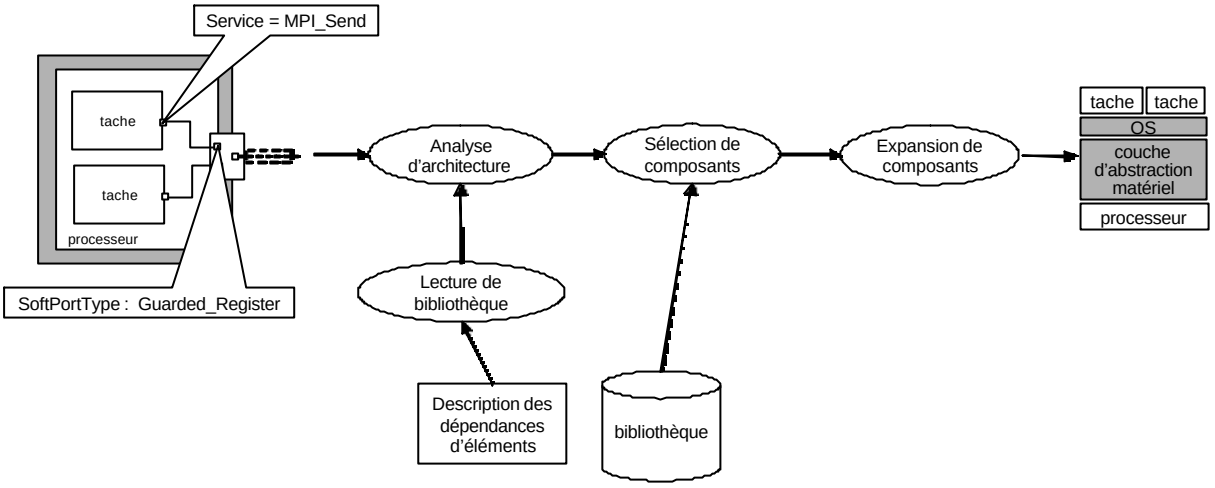


Figure 46 ASOG

La Figure 46 montre les quatre actions exécutées par ASOG pour générer l'interface logicielle : analyse de l'architecture, lecture de la bibliothèque, sélection des composants, et expansion des composants. A l'étape analyse de l'architecture, l'outil recherche le matériel utilisé (le processeur), les services requis (paramètres des ports de tâche), et l'implémentation de ce service (paramètres des ports internes du processeur). Et puis, l'outil sélectionne les éléments de bibliothèque nécessaires pour construire ces services. Enfin, les éléments de bibliothèque sont paramétrés pour obtenir le code de l'interface logicielle.

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

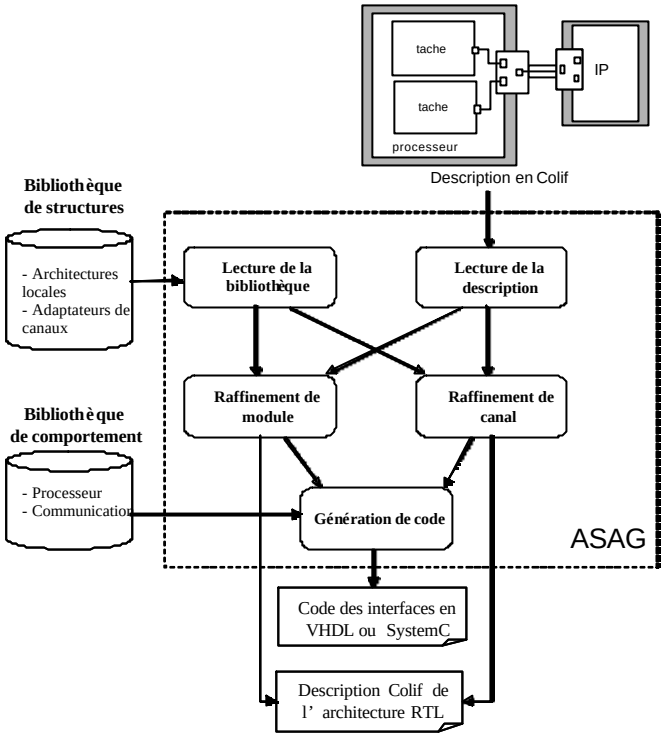


Figure 47 Génération des interfaces matérielles par ASAG

L'outil CosimX est utilisé pour valider les systèmes décrits à plusieurs niveaux d'abstraction. Il permet de rendre exécutable la description VADeL du système. L'exécution du système avant génération des interfaces permet de valider le comportement des tâches, des IP ainsi que le choix des services de communication. CosimX offre aussi la possibilité de tester le fonctionnement du système après le raffinement partiel ou complet du système (co-simulation).

5.3.2 L'automatisation de l'adaptation

Après le développement des composants et la génération des interfaces, on peut commencer le prototypage. Dans cette phase, les concepteurs possèdent toutes les parties de l'application au niveau RTL. Les parties matérielles sont décrites en langage de description de matériel. Elles sont synthétisables. Les parties logicielles sont en langage C/C++ ou en langage assembleur.

Les parties logicielles ne correspondent pas encore à la version finale (celle implémentée sur la puce), car leur validation est faite sur la plateforme de prototypage, puis sur la puce finale. Mais ce prototypage suppose de repenser les couches du logiciel dépendant du matériel, ce qui constitue l'adaptation de l'application à la plateforme.

La modification manuelle de programmes est une tache difficile. Mais l'outil ASOG capable de générer un système d'exploitation minimal et la couche d'abstraction du matériel peut être utilisé pour régénérer une couche d'abstraction du matériel adaptée à la plateforme. Cette transformation peut être vue comme une abstraction des couches logicielles de bas niveau à extraire de l'architecture RTL de l'application pour permettre une nouvelle génération. Cette abstraction est relativement facile dans notre cas puisque ces couches logicielles sont déjà le résultat d'une génération automatique ASOG. Ce qui est fait, est de revenir en arrière dans le flot de conception, et de changer les paramètres pour obtenir les couches basses du logiciel adaptée à la plateforme en conservant les même services appelés par les taches. Comme on l'a déjà mentionné à plusieurs reprises, cette phase d'adaptation est difficile à faire manuellement. Et enfin, générer du code automatiquement pour ensuite le modifier à la main est une perte de temps qui pourrai être évitée. Cette nouvelle génération doit être transparente du point de vue du logiciel de l'application, et les concepteurs ne doivent pas voir les différences d'implémentation de la partie matérielle.

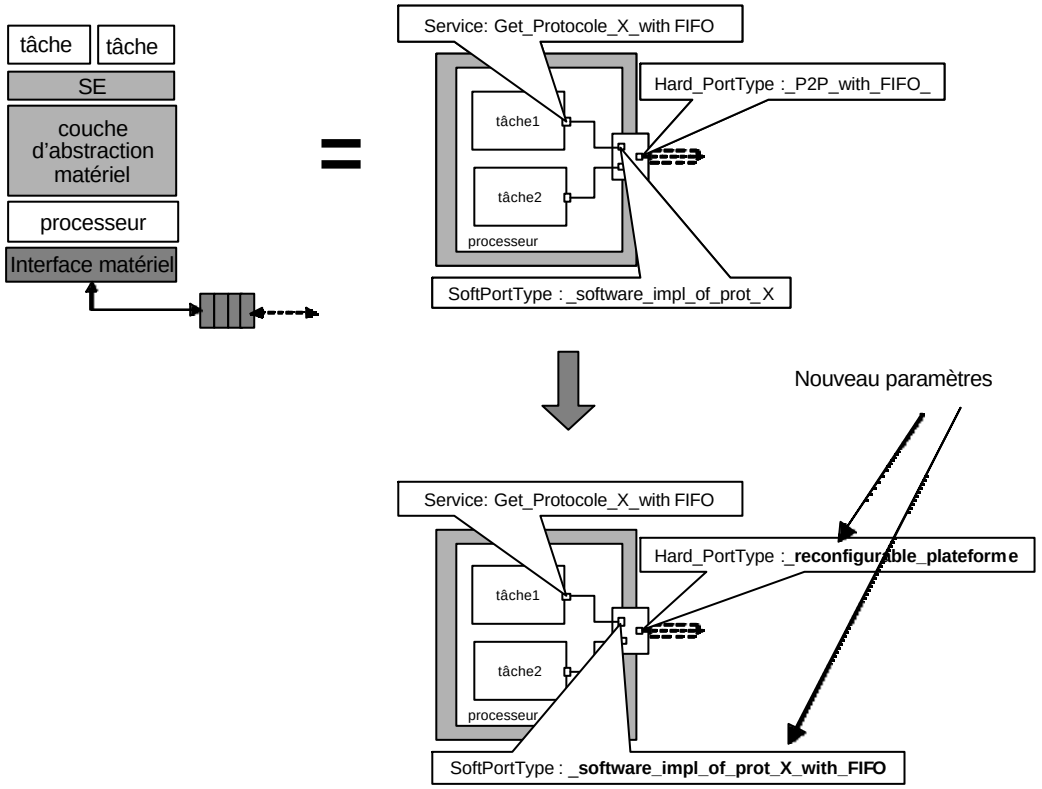


Figure 48 Abstraction et modification de paramètres

La Figure 48 illustre un exemple de cette adaptation. Dans cet exemple, un appel de service *get_protocole_X_with FIFO* est requis par une tache. Cet appel est une demande de

donnée au port qui utilise un protocole X. Ce port est implémenté en logiciel et en matériel. Le paramètre *SoftPortType* dans le port interne du processeur indique l'implémentation logicielle de ce protocole. Le paramètre *HardPortType* dans le port externe du processeur indique une réalisation matérielle de l'interface. Dans ce paramètre, on trouve que ce port est implémenté en utilisant une connexion point-à-point et une FIFO.

Pour avoir une nouvelle couche d'abstraction du matériel, on modifie les paramètres des ports interne et externe du processeur en gardant le même paramètre de port de tache. Dans cet exemple, on garde le paramètre de service *get_protocole_X_with FIFO* dans le port de la tache, mais les implémentations logiciel/matériel sont changées. On n'utilise plus une connexion point-à-point avec une file (FIFO), on utilise le réseau de communication disponible dans la plateforme reconfigurable. L'implémentation de la FIFO est déplacée en logiciel. Le logiciel doit alors abstraire l'implémentation du matériel et émuler la FIFO matérielle en mémoire.

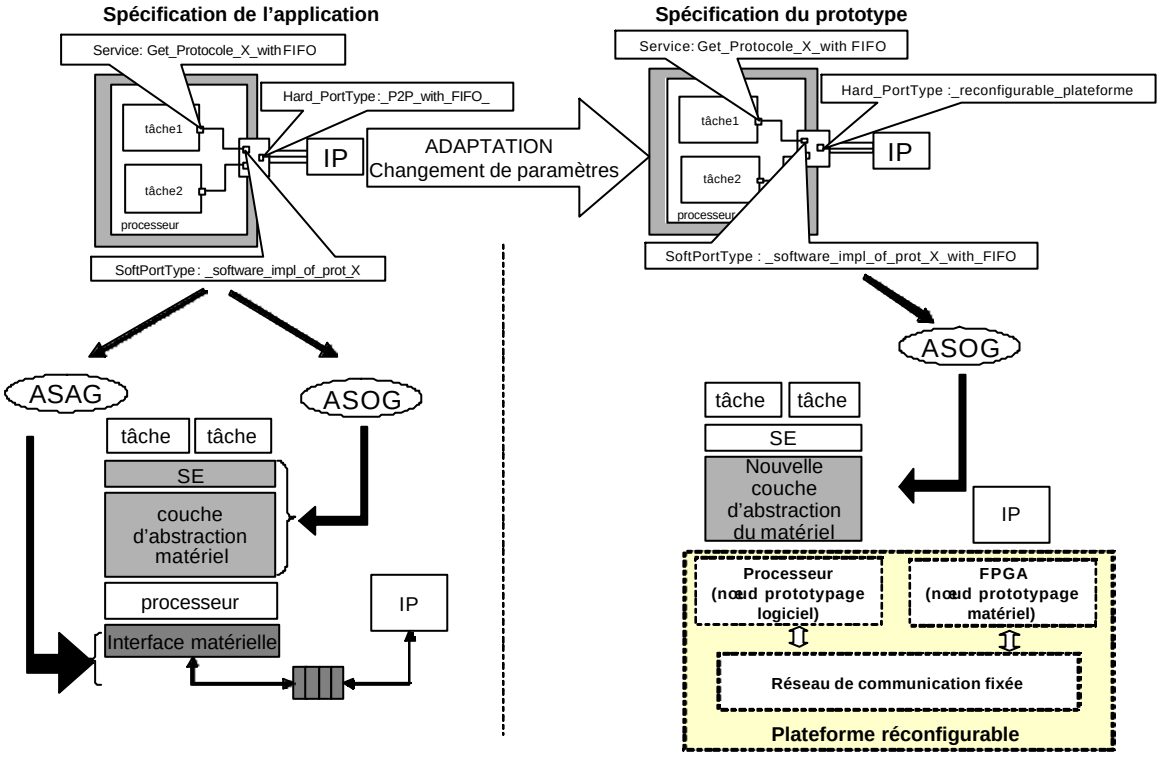


Figure 49 Génération de la couche d'abstraction du matériel

La Figure 49 montre la génération de la nouvelle couche d'abstraction du matériel pour le prototype. On génère une nouvelle couche d'abstraction du matériel à partir d'un nouveau modèle (modèle de haut niveau adapté). Avec cette nouvelle couche, le logiciel pilote le

matériel de la plateforme en utilisant le même appel de service. En plus, cette couche émule la fonctionnalité de la partie matérielle qu'on ne peut pas réaliser sur la plateforme. Dans cet exemple, la FIFO matérielle est émulée par cette couche.

5.3.3 Avantages et limitations

5.3.3.1 Accélération du processus du prototypage

Le prototype développé dans la paragraphe précèdent permet d'accélérer l'adaptation grâce à l'utilisation d'outil de génération automatique ASOG. On peut donc obtenir plus rapidement le prototype. De plus, la génération automatique permet la réutilisation des éléments de bibliothèque. Ils ne sont alors développés qu'une seule fois.

La réutilisation et la génération automatique permettent aussi d'éviter des erreurs. Si la bibliothèque est correcte, on a juste besoin de modifier les paramètres pour générer la bonne couche d'abstraction du matériel.

5.3.3.2 Nécessité de la bibliothèque

Pour utiliser cette technique, la bibliothèque doit avoir les éléments pour réaliser tous les services requis sur la plateforme. Ces éléments doivent avoir les mêmes comportements que sur la puce finale.

L'utilisation d'élément de bibliothèque est décrite sur [GAU01]. A partir d'un service requis par une tâche, l'outil ASOG détermine l'élément utilisé. Il est possible d'avoir plusieurs réalisations de cet élément. Le outil ASOG décide alors de choisir une réalisation avec les bons paramètres. Ce choix et la valeur des paramètres dépendent des paramètres associés aux ports interne et externe.

Chaque élément doit posséder dans la bibliothèque au moins deux solutions de réalisation, l'une pour le système monopuce, l'autre pour la plateforme. En pratique, il y en a plus car un élément ou un service peut être découpé en parties logicielle et matérielle avec des frontières différentes [PAV04]. Il faut aussi s'assurer de la validation de ces éléments de bibliothèque et leur compatibilité, ce qui reste une difficulté non abordée dans cette thèse.

5.3.3.3 Limitation des possibilités de validation

L'intérêt d'une adaptation automatique est d'obtenir au plus vite le prototype pour développer et valider ou optimiser au plus tôt les parties logicielles. La Figure 50 montre

qu'une couche d'abstraction du matériel est nécessaire pour exécuter le logiciel à développer sur la plateforme.

L'exécution du logiciel d'application sur ce prototype présente des avantages par rapport à la co-simulation. La co-simulation en utilisant un ISS et un simulateur (VHDL par exemple) est très lent. Il est alors difficile de vérifier le logiciel d'application, souvent très complexe et dont l'exécution peut prendre beaucoup de temps. La plateforme permet d'observer les vrais effets du parallélisme et de valider la synchronisation entre les tâches.

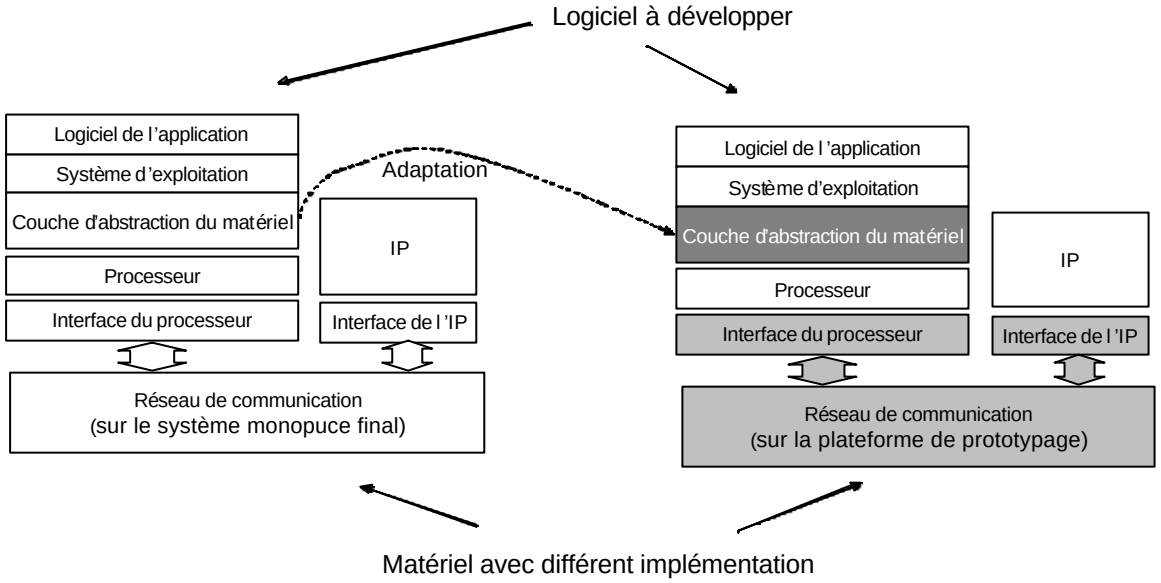


Figure 50 L'adaptation pour développer le logiciel

On peut aussi vérifier les parties du système d'exploitation qui sont complètement indépendantes de matériel : l'ordonnanceur, les sémaphores, la pile, …… Mais, ce qui dépend du matériel est difficile, voire même impossible à vérifier : le latence d'une d'interruption, compteur de temps, …. .

Le prototypage tel que décrit dans cette thèse ne permet pas le développement ou la validation de la couche logicielle d'abstraction du matériel.

5.3.3.4 Nécessité d'un bon découpage en couches logicielles

Cette technique d'adaptation repose sur le principe que la partie logicielle de l'application est structurée en couches bien différenciées. Dans ce cas, seule la partie dépendante du matériel est modifiée, ce qui laisse la partie applicative et le système d'exploitation identiques sur le prototype et sur l'application finale.

5.3.4 Conclusion sur l'adaptation

Dans ce paragraphe, on a discuté d'une méthode de prototypage sur une plateforme reconfigurable qui permet d'avancer le développement du logiciel. Afin d'obtenir rapidement un prototype, on utilise le flot ROSES de l'équipe SLS pour automatiser l'étape d'adaptation. Ce flot permet de générer une couche d'abstraction du matériel. Cette couche est utilisée pour remplacer celle qui n'est pas compatible avec la plateforme. On peut alors prototyper l'application sur cette plateforme reconfigurable. Ce prototype est alors utilisé comme un environnement pour exécuter le logiciel d'application. Le développement du logiciel le plus tôt possible écourte le temps de conception global des systèmes monopuces car les concepteurs développent le logiciel sur un environnement de développement presque similaire à la puce finale.

5.4 Co-émulation

5.4.1 Exemple de co-émulation en utilisant la plateforme ARM Integrator[1]

Un moyen pour réaliser un prototype est la co-émulation. La co-émulation consiste à mélanger la simulation et l'émulation pour prendre les avantages des deux techniques de vérification. La simulation donne une bonne observabilité et des possibilités d'utiliser des niveaux d'abstraction élevés. L'émulation permet une exécution rapide.

Dans ce paragraphe, nous allons faire une expérience de co-émulation en utilisant la plateforme ARM Integrator. Comme exemple de l'application, nous utilisons l'application DivX expliquée dans le paragraphe 4.4.1.

5.4.1.1 Les principes

Avant d'aborder la co-émulation, je vais discuter un peu de co-simulation en utilisant SystemC puisque notre expérimentation est basée sur le même principe. Dans le schéma de principe de la Figure 51, le module SystemC de plus haut niveau encapsule plusieurs sous-modules SystemC (représentant le comportement de l'application) et un module non SystemC, appelé adaptateur SystemC. Ce module est une interface avec un autre simulateur

[1] Un travail commun avec M. Jinyong Jung

(simulateur VHDL, ISS, …) appelé pour simuler une partie du système. Ce module adaptateur SystemC communique avec le simulateur en utilisant une mémoire partagée.

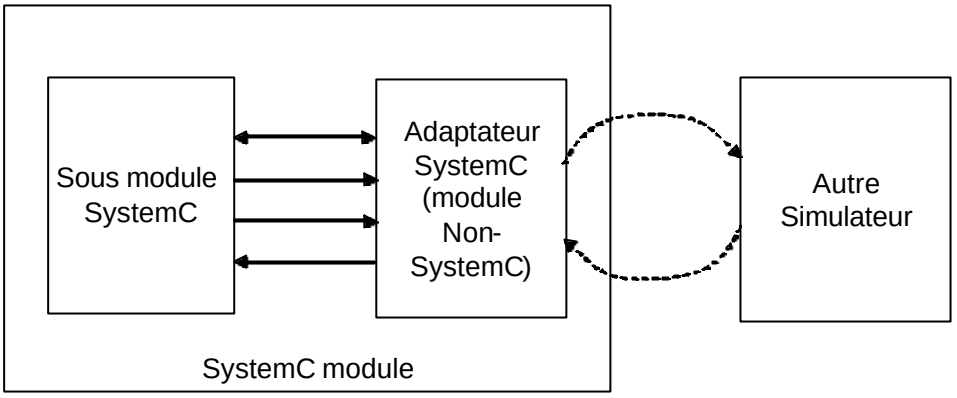


Figure 51 Co-simulation avec SystemC

Ce même principe est utilisé pour la co-émulation. L'adaptateur SystemC communique avec une interface de co-émulation qui elle est connectée à l'émulateur (Figure 52). L'interface de co-émulation est un pilote (logiciel) qui gère la communication physique avec la plateforme. Cette interface est spécifique selon, le type de connexion mis en œuvre (port série, USB,….) et le système d'exploitation (Windows, Linux,…).

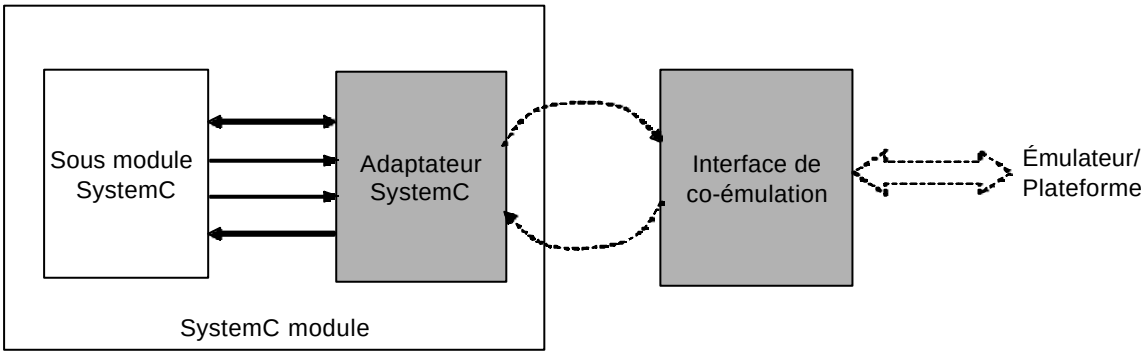


Figure 52 Connexion entre une simulation SystemC et un émulateur

5.4.1.2 Connections physiques avec l'ARM Integrator

Pour la co-émulation, on a besoin d'établir une connexion entre l'ordinateur et la plateforme que l'on utilise. Nous analysons cinq solutions possibles de connexions physiques que l'on peut utiliser sur ces plateforme. Le choix des connexions est très dépendant de la plateforme utilisée.

5.4.1.2.1 Connexion multi-ICE

Le premier choix est le multi-ICE et les interfaces JTAG sur la carte processeur et la carte FPGA (Figure 53). On connecte l'environnement de simulation SystemC au serveur multi-ICE à travers l'interface de co-émulation. Cette interface de co-émulation agit comme client de multi-ICE.

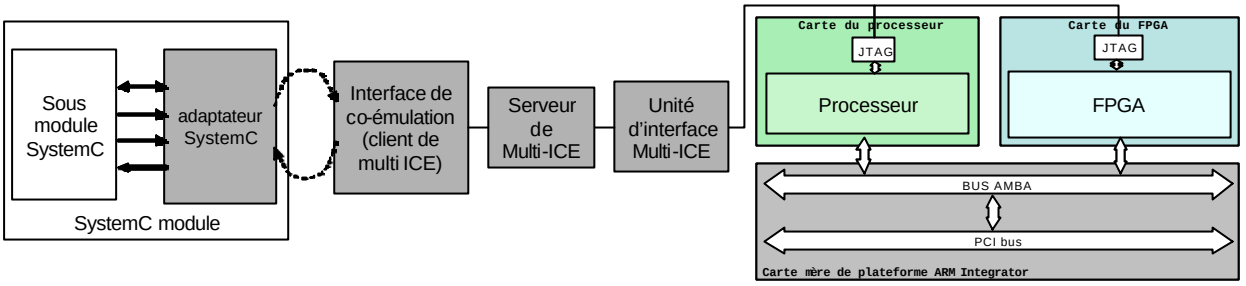


Figure 53 Co-émulation en utilisant une connexion multi-ICE

La partie à développer dans ce schéma est l'interface de co-émulation (client de multi-ICE) et un contrôleur JTAG pour la carte FPGA. Le goulot d'étranglement de cette connexion est la vitesse du JTAG sur multi-ICE qui est 20 K bits par seconde.

5.4.1.2.2 Connexion en utilisant de carte réseau

Le deuxième choix est d'utiliser d'une carte réseau (carte Ethernet). On utilise des cartes réseau des deux cotés : sur l'ordinateur et sur la plateforme. La carte réseau est insérée sur le bus PCI de la plateforme. Le processeur ARM et la carte FPGA communiquent alors avec la carte réseau à travers le bus AMBA et bus PCI (Figure 54). La connexion entre les deux cartes réseau est réalisée à travers un réseau local d'ordinateurs (LAN : local area network).

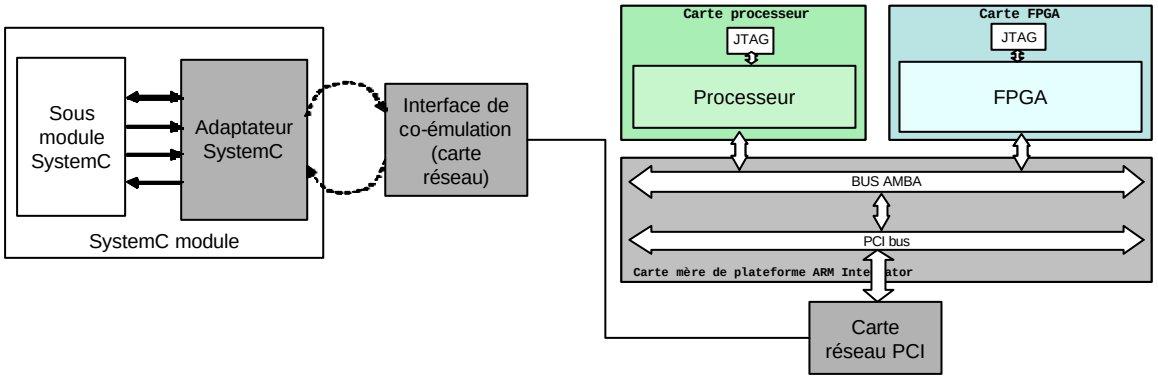


Figure 54 Co-émulation en utilisant une carte réseau

Pour réaliser cette configuration, on doit développer les fonctions du protocole TCP/IP (protocole pour internet/réseau), et un pilote pour la carte réseau sur la plateforme. La vitesse maximum de cette connexion est limitée à 10 Mbits/s.

5.4.1.2.3 Connexion en utilisant un pont PCI/PCI

Une alternative de connexion que l'on peut développer est un pont de bus PCI/PCI. L'ordinateur et la plateforme possèdent des bus PCI et des connecteurs PCI (PCI slot). On peut donc les connecter directement à travers un pont. Cette connexion est inhabituelle car on a deux hôtes : le processeur de la plateforme et l'ordinateur.

Pour connecter deux bus PCI (celui de la plateforme et celui du PC), il faut développer une carte PCI/PCI et les fonctions associées à ce protocole pour les processeurs et les FPGA de la plateforme, et dans l'interface de co-émulation. La vitesse de cette connexion est limitée à 133 Mbits/s.

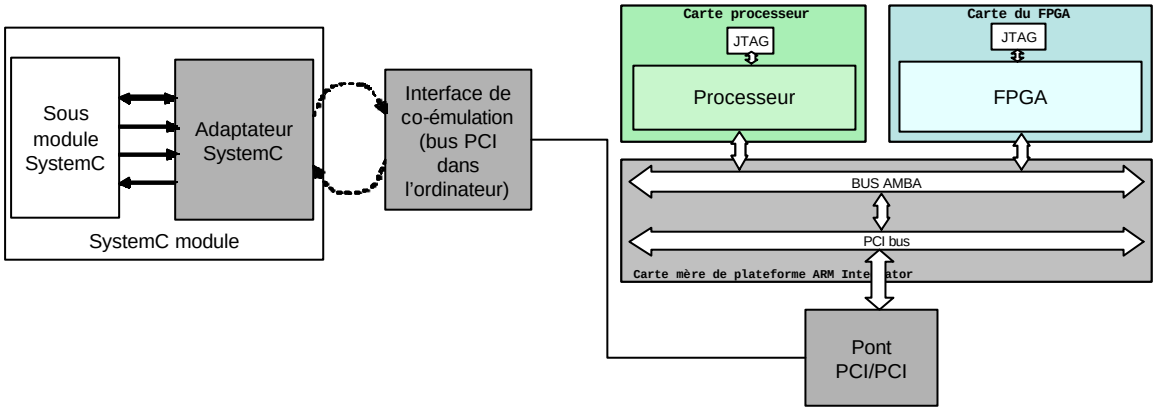


Figure 55 Co-émulation en utilisant un pont PCI/PCI

5.4.1.2.4 Connexion en utilisant une carte USB PCI

Une autre solution est d'utiliser une carte USB PCI (une carte USB insérée sur le connecteur PCI) pour connecter la plateforme à l'environnement de simulation (l'ordinateur).

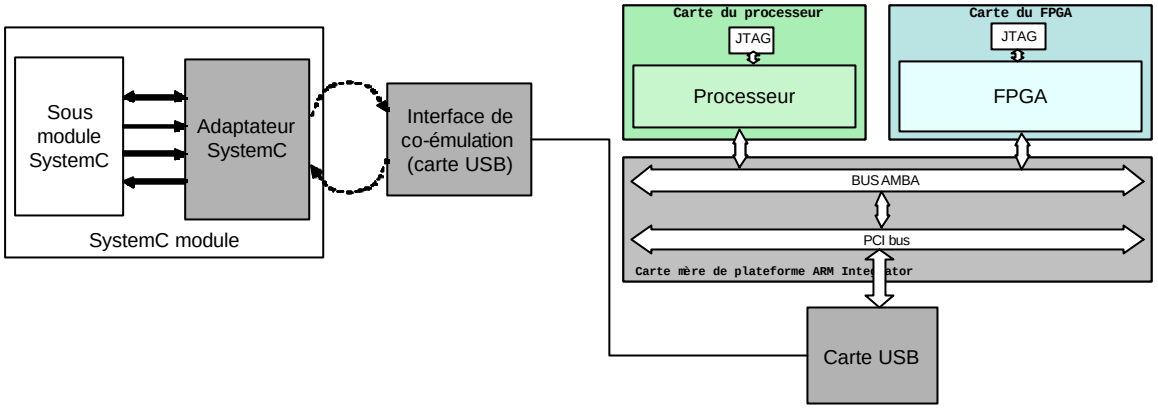


Figure 56 Co-émulation en utilisant une connexion USB

Pour réaliser cette co-émulation, on ne doit développer que le pilote de la carte USB [BRO03] sur la plateforme. Ce pilote sera réalisé en logiciel et en matériel. Le pilote logiciel est utilisé par le processeur, et le pilote matériel est utilisé par le FPGA. La vitesse maximum de cette connexion et 6 Mbit/s.

C'est cette solution qui sera mise en œuvre pour des raison de simplicité et de performances.

5.4.1.3 Cas d'étude avec l'application VDSL

Dans cet expérience, nous utilisons le même application avec la section précèdent : VDSL avec le même architecture. Dans cette expérimentation, un seul des processeurs s'exécute sur la plateforme ARM Integrator pendent que l'on simule les autres parties du système dans un ordinateur en utilisant systemC. Dans la simulation systemC, la partie émulée est représentée par un module systemC appelé Adaptateur SystemC. Ce module communique avec le processeur sur la plateforme à travers une connexion USB. Afin d'envoyer et de recevoir des données de la carte USB, cet adaptateur utilise des appels de fonctions de pilote USB. La carte USB de l'ordinateur est reliée à la carte USB PCI, elle-même connectée sur la plateforme ARM Integrator. Pour envoyer et recevoir des données, cette carte doit alors traverser le bus PCI, le pont de PCI/AMBA-AHB, et bus AHB. Avant arriver au tâches de l'application, les données sont décodés/encodés par une couche d'E/S. Cette couche initialise la connexion USB et gère les échanges de données entre le processeur et la partie émulée. La Figure 57 illustre cette expérimentation.

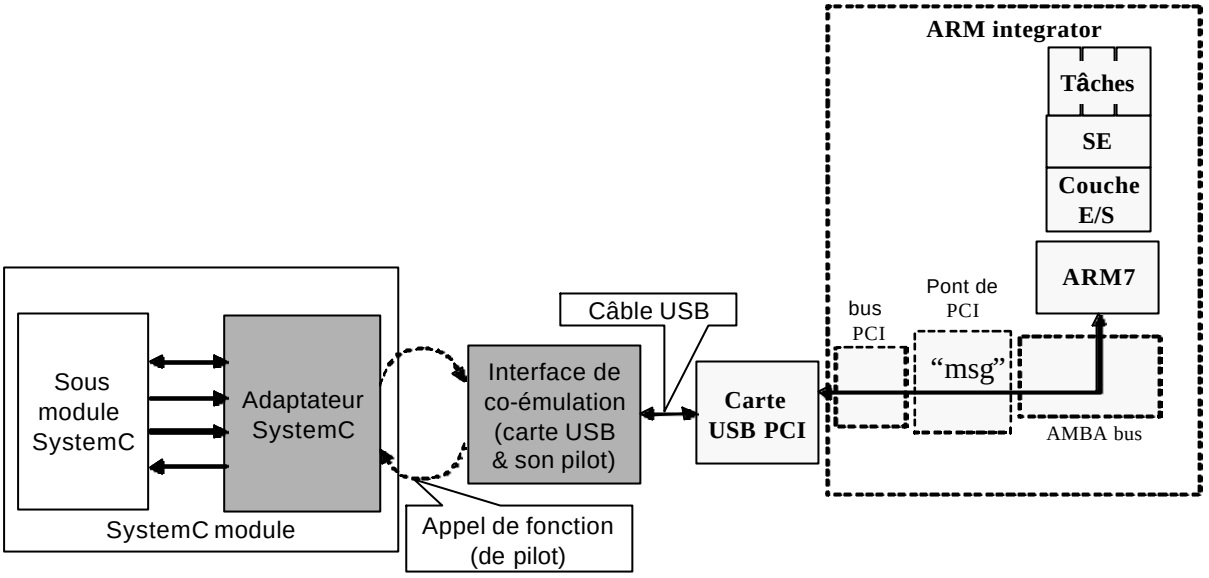


Figure 57 L'expérimentation sur co-émulation

Il existe deux types de communication entre la plateforme et la partie simulée :

- la partie émulé initie une communication avec la partie simulée
- la partie simulée initie une communication avec la partie émulée

Quand la partie émulée souhaite communiquer avec la partie simulée, c'est-à-dire, quand le processeur veut envoyer ou prendre des donnés, il s'effectue les actions suivantes :

- Pour envoyer les données, le processeur formate les données (couche logicielle de bas niveau), les écrits sur bus AMBA vers la carte USB/PCI. Ces informations sont récupérées par le simulateur qui demande à la carte USB de générer une interruption à l'attention du processeur pour acquitter de la réception. Pendant ce temps, le processeur attend l'acquittement. Cette interruption ne fait pas partie de l'application, seulement du protocole de communication entre le simulateur et l'émulateur.
- Pour recevoir une donnée, le principe est le même, mais le simulateur envoie les données et demande à la carte USB de générer une interruption. A la réception de l'interruption, le processeur lit les donnés sur le bus AMBA.

Quand la partie simulée souhaite communiquer avec la partie émulée, il faut envoyer une interruption au processeur émulé. Le traitement de cette interruption se fait par une couche logicielle de bas niveau spécifique. On vérifie par exemple si l'interruption reçue est

une interruption de l'application ou un acquittement. Si l'interruption est une interruption de l'application, le processeur exécute le traitement de l'interruption issue de l'application.

5.4.1.4 Difficultés, résultats, et analyse

Dans cette expérimentation, on obtient une vitesse d'échange de données de 8 Kbits/s. Cette vitesse dépend de l'application, plus précisément, de la taille des données envoyée. Cette vitesse est beaucoup plus faible que la vitesse de connexion USB à cause de la latence introduit par la couche E/S et l'interface de co-émulation. Malgré tout, le goulot d'étranglement de cette co-émulation est sur la partie simulée.

En utilisant cette co-émulation, on peut vérifier plus facilement la partie matérielle simulée sur l'ordinateur. Cette vérification est plus facile à faire que si l'on réalise la partie matérielle avec un FPGA car tous les signaux de matériel simulé sont là. On n'a pas besoins de sélectionner et de registrer quelques signaux avant l'exécution en utilisant analyseur logique. De plus, on ne doit pas réaliser de circuit de déboguage pour observer des signaux à l'intérieur de module.

5.4.2 Evaluation

La co-émulation combine plusieurs techniques de vérification pour avoir les avantages de chaque technique. On peut considérer la co-émulation comme un prototypage avec quelques composants simulés.

En co-émulation, il y a deux sujets à aborder : la communication et la synchronisation. La communication est abordée dans la partie perspective qui suit ce paragraphe. En effet, de part la nature de l'application et l'objectif de prototypage, une synchronisation par échanges de données convenait parfaitement. Dans l'expérimentation de co-émulation, on explore plusieurs solutions sur le sujet de communication. On trouve trois problèmes à résoudre sur ce sujet : (1) définir la connexion physique entre le simulateur (ordinateur) et la plateforme, (2) développer une couche logicielle entre le simulateur et la connexion physique (dans notre exemple : adaptateur SystemC), et (3) définir une couche logicielle entre le logiciel exécuté sur la plateforme et la connexion physique (dans notre exemple : la couche d'E/S).

La connexion physique choisie est une liaison USB. 2.0, pour une raison de simplicité de mise en œuvre, un coût extrêmement réduit (achat une carte PCI/USB) et une bande passante très supérieure à ce que peut fournir un simulateur. Dans la connexion USB, on a

deux cotés différents : un hôte USB et un client USB (anglais : *USB host* et *USB device*). Le coté hôte prendre le contrôle de la communication entre deux cartes USB. Dans l'expérience, l'ordinateur s'agit comme coté hôte, et la plateforme agit comme la coté client. On utilise alors une carte USB/PCI normale sur l'ordinateur, mais on utilise une carte PCI/USB Net 2280 de Netchip [NET03]. Cette carte est conçue pour le développement de client USB. Cette carte alors peut être configurée comme un client USB (Une carte USB normale agit toujours comme hôte). Cette configuration est prise pour faciliter le développement de couches logicielles sur le simulateur (interface de co-émulation) et sur le processeur ARM.

La couche logicielle entre le simulateur et la connexion physique consistait à réécrire des pilotes USB sous Linux en partant de pilotes existants et trouvés sur Internet [BRO03]. En utilisant ces pilotes, l'adaptateur SystemC configure la communication entre les deux cartes USB et aussi envoie et reçoit des données codées (des messages). De plus, on peut utiliser quelques fonctions fournies avec la carte USB Netchip utilisée pour le développement.

Sur le processeur ARM, la couche E/S consiste de deux parties. La première partie est le pilote de la connexion, et la deuxième est la partie qui effectue le codage et le décodage. Le pilote de connexion configure la connexion entre processeur ARM et la carte USB. Etablir la communication n'est pas très facile. D'abord, il faut configurer le pont AMBA/PCI, et puis configurer la carte PCI/USB. On utilise une partie des pilotes existants fournis avec la carte USB/PCI et un exemple de configuration de bus PCI. La deuxième partie effectue le codage et le décodage de données. Il y deux types de communication. Le première type est quand le processeur envoie ou reçoit des données. La deuxième type est quand la partie simulée (le sous module systemC) envoie une interruption au processeur ARM. Cette deuxième partie de couche E/S s'occupe les deux types de communication.

5.4.3 Perspectives

Pour aller plus loin sur la co-émulation, on peut avoir plusieurs techniques de synchronisation entre la partie simulée et la plateforme : synchronisation basée sur des événements discrets, synchronisation basée sur des délais annotés, synchronisation au cycle près. La précision et la vitesse de la co-émulation dépendent de cette technique de synchronisation.

Dans la synchronisation basée sur des événements discrets, l'environnement de simulation et la plateforme s'échangent les donnée seulement quand il y a des événements tels que des interruptions ou écritures/lectures des données. On ne peut que vérifier la fonctionnalité avec ce type de co-émulation, mais avec une vitesse élevée.

Dans la synchronisation basée sur des délais annotés, on utilise un délai prédéterminé pour la synchronisation. Dans ce cas là, on doit fixer le délai avant l'exécution.

Dans la synchronisation au cycle près, les échanges des données sont effectués à chaque cycle. Afin de les faire, le simulateur et la plateforme doivent s'exécuter cycle par cycle. Cette co-émulation est très précise, mais aussi très lente.

CHAPITRE 6 : CONCLUSION

La technologie microélectronique facilite l'intégration de nombreux composants sur une même puce, ce qui permet d'atteindre les performances et de répondre aux besoins de fonctionnalité exigés par les applications. La tendance dans la conception de tels systèmes, appelés systèmes monopuces, est à l'augmentation de la complexité. On observe une très grande hétérogénéité des composants, spécifiques pour certaines fonctionnalités, ou programmables (processeur). La partie logicielle devient elle aussi très complexe et requière une organisation sous forme de couches pour assurer la portabilité du programme de l'application.

Les applications utilisant ces systèmes monopuces sont sur un marché très concurrentiel, et l'arrivée rapide du produit sur le marché est très importante. Ceci entraîne une énorme pression pour réduire le temps de conception et de validation. De plus, le coût lié à la conception des parties matérielles et logicielles est très élevé. Et détecter une erreur après fabrication entraîne un surcoût financier et de temps non acceptable.

Ceci nous entraîne vers les deux problèmes explorés dans ce travail de thèse : s'assurer que la système est correct avant sa fabrication et accélérer le processus de conception.

Après d'avoir évalué plusieurs techniques de vérification, nous pensons que le prototypage sur une plateforme reconfigurable est une bonne solution pour résoudre les problèmes mentionnés. Ce type de prototypage permet de vérifier rigoureusement les systèmes grâce à la vitesse élevée, et de tester le système dans son environnement physique d'utilisation. Il accélère aussi le processus de conception en permettent le développement de certaines couches logicielles avant que le système soit près. Dans ce cas, le prototype est une plateforme d'exécution pour le logiciel développé.

Pour obtenir rapidement un prototype à partir d'une description RTL d'une application, nous proposons un flot de prototypage basé sur une plateforme reconfigurable et composé de quatre étapes. Ces étapes sont : allocation, configuration de la plateforme, adaptation de l'application, et génération du code.

Dans **l'allocation**, les concepteurs associent chaque partie de l'architecture à un nœud de prototypage de la plateforme. Ces associations indiquent sur quelles parties de la plateforme reconfigurable sont réalisées les parties de l'architecture de l'application. **La configuration** est la réorganisation de la plateforme reconfigurable. **L'adaptation** consiste à modifier l'application pour satisfaire aux caractéristiques de la plateforme reconfigurable. Cette étape est effectuée si la plateforme ne peut pas être configurée pour s'adapter aux besoins de l'application. Enfin, la dernière étape est **la génération du code**. Cette étape consiste en processus standard tels que la compilation et l'édition de lien des logiciels, la synthèse logique, le placement sur FPGA, et le routage.

Ce flot est validé en réalisant le prototypage des applications VDSL et DivX. La plateforme utilisée est une plateforme ARM Integrator avec une carte mère, quatre modules processeurs ARM, et un module FPGA. La communication entre les modules se fait à travers un bus AMBA AHB.

Le développement d'une application VDSL multiprocesseur avait pour objectif de valider les interfaces logicielles/matérielles générées automatiquement par l'outil ROSES du groupe SLS. Pour cette application, la configuration de la plateforme était nécessaire pour contourner le problème du bus AMBA fixé par la plateforme. Une adaptation de l'application était aussi nécessaire pour prendre en compte l'espace adressage de la plateforme différent de celui du VDSL. L'adaptation ne concerne que la couche logicielle la plus basse, dite couche d'abstraction du matériel. Ce prototype a été utilisé pour valider les interfaces matérielles et le système d'exploitation.

Une application DivX est utilisée dans la deuxième expérience. L'architecture de l'application consiste en quatre processeurs qui communiquent en utilisant un protocole MPI réalisé en logiciel et en matériel. L'objectif de cette expérience est d'obtenir une plateforme pour valider le logiciel de l'application. La difficulté du prototypage repose sur la différence de réalisation du protocole MPI dans le système final et sur la plateforme. En effet, l'architecture de la plateforme ne permet pas qu'une partie de ce mécanisme de communication soit réalisé en matériel. Il faut donc le faire entièrement en logiciel, ce qui doit être invisible du point de vue de l'application. Dans cette expérience, l'adaptation était une partie critique et délicate, consistant à modifier la couche logicielle d'abstraction du matériel. Cette expérience a permis de valider les tâches de l'application sur les processeurs.

L'adaptation consiste donc principalement à modifier la couche logicielle d'abstraction du matériel. Comme cette couche logicielle d'abstraction est obtenue automatiquement à l'aide de l'outil ASOG développé dans le groupe SLS, il nous semblait possible d'automatiser l'étape d'adaptation, en modifiant non pas la couche logicielle, mais l'entrée de l'outil ASOG. Cet outil étant basé sur une bibliothèque de fonctions, il a été nécessaire de développer de nombreux éléments.

En addition des deux expériences de prototypage, une expérience de co-émulation a été faite pour explorer les difficultés et les avantages de cette technique. L'avantage principal est qu'on peut profiter de l'observabilité de la simulation et de la vitesse de l'émulation. De plus, la co-émulation permet de réaliser un prototype même si on n'a pas tous les composants. Il y a deux problèmes techniques liés à la co-émulation : la communication et la synchronisation entre la partie simulée et la partie émulée. Une solution matérielle et logicielle est proposée dans cette thèse pour la partie communication. Mais la partie synchronisation n'est pas abordée.

Ce travail de thèse montre que l'on peut obtenir rapidement un prototype en utilisant le flot proposé sur une plateforme reconfigurable. Un tel prototype est très utile pour valider de nombreux aspects de la conception : interfaces de communication, système d'exploitation, partie matériel, logiciel de l'application, … . L'avantage d'une plateforme reconfigurable est aussi de faciliter le développement des parties logicielles, donc accélérer la conception.

Ce travail n'est qu'un début d'exploration des possibilités offertes par le prototypage pour valider les systèmes monopuces. Il reste de nombreux points à étudier ou à approfondir. Concernant les plateformes de prototypage, le point faible semble être le manque de configurabilité de la partie communication pour mettre en œuvre différents mécanismes et protocoles de communication entre les nœuds de prototypage. Il faut aussi penser à intégrer ce flot de prototypage dans le flot classique de conception de systèmes monopuces. En effet, on constate que tous les flots de prototypage existants sont des flots séparés qui introduisent une rupture dans la conception. A noter aussi que le prototypage requiert l'utilisation de nombreux outils (chaîne de compilation, partitionnement, synthèse, placement routage) ce qui ne facilite pas l'intégration.

La co-émulation semble être un début de réponse aux deux problèmes décrits précédemment. La partie simulation d'un système co-émulé permet de modéliser des

mécanismes de communication de complexes, et offre en plus une grande flexibilité sur le modèle. Et cette partie simulation fait le lien avec le flot de conception classique ou la simulation est la principale technique de vérification. On peut alors travailler de façon incrémentale en déplaçant les composants de la simulation vers la plateforme en fonction de l'avancement de la conception.

BIBLIOGRAPHIE

[ADA98] Adaies K., Alexiou G.P., Kanopoulos N.,"An Extensible, Low Cost Rapid Prototyping Environment based on a Reconfigurable Set of FPGA" 9th Rapid System Prototyping (RSP), Leuven, 1998

[APT02] Aptix Corp., "Prototype Studio: RTL to PSP Pre Silicon Prototypes for System-on-chip Designs", http://www.aptix.com/products/product_overview.htm., Aptix Corp., 2002

[ARM99a] ARM Limited, "AMBA Specification (Rev 2.0)", ARM Limited, 1999

[ARM99b] ARM Ltd., "Integrator/AP User Guide", ARM Limited, 1999

[ARM99c] ARM Limited, "Integrator/LM-XCV600E+ Integrator/LM-EP20K600E+ User Guide", ARM Limited, 1999

[ARM99d] ARM Limited,"Integrator/CM7TDMI User Guide", ARM Limited, 1999

[ARM99e] ARM Limited, "ARM Developer Suite AXD and armsd Debuggers Guide", ARM Limited, 1999

[ARM99f] ARM Limited, "Multi-ICE User Guide", ARM Limited, 1999

[ARM02] http://www.arm.com/armtech/primeXsys

[BRO03] David Brownell, "Linux and USB 2.0", http://linux-usb.org/usb3.html, 2003

[CAR98] Cardelli S., Chiodo M., Giusto P., Jurecska A., Lavagno L., Sangiovanni-Vincentelli A., "Rapid-Prototyping of Embedded System via Reprogrammable Device", dans DAES volume 3, pages : 149-161, Kluwer Academic Publisher, 1998

[CES01] Cesario W., Nicolescu G., Gauthier L., Lyonnard D., "Colif: a Multi Design Representation for Application-Specific Multiprocessor System-on-Chip Design", 12th Rapid System Prototyping (RSP), California, 2001

[CES02] W. Cesário, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, “Component-Based Design Approach for Multicore SoCs”, Design Automation Conference (DAC), New Orleans, 2002

[CHA99] Chang H., Cooke L., Hunt M, Martin G, McNelly A, Todd L., “Surviving the SOC Revolution – A Guide to Plateforme-Based Design”, Kluwer Academic Publisher, 1999

[CLO02] Clouard A., Mastrorocco G., Carbognani F., Perrin A., Ghenassia F., “Toward Bridging the Precision Gap between SoC Transactional and Cycle-Accurate Levels”, Design automation and Test in Europe (DATE), Paris, 2002

[DAL00] Mitchel Dale, “The Value of the Hardware Emulation”, http://www.mentor.com/emulation, Mentor Graphics, 2000

[DIV01] Project Mayo, http://www.projectmayo.com, 2001

[DOR01] Dorfel M., Hofmann R.,"A Prototyping System for High Performance Communication System" ,12th Rapid System Prototyping (RSP), Leuven, 2001

[EVA03] Evans Data Corporation, ”Embedded Systems Development Survey, Volume 1, 2003”, http://www.evansdata.com/n2c/surveys/embedded_toc_03_2.shtml, 2003

[ROS98] Wolfgang Rosentiel, “Rapid Prototyping, Emulation and Hw/Sw Co-debugging”, une chapitre dans : A. A. Jerraya et J. Mermet, “System Level Synthesis”, pages : 219-262, Kluwer Academic Publisher, 1998

[HAR97a] Hardt W., Rosentiel W., “Prototyping of Tightly Coupled Hardware/Software System”, dans DAES volume 2, pages : 283-317, Kluwer Academic Publisher, 1997

[HAR97b] Hartenstein R.W., Becker J., Herz M., Nageldinger U., "An Innovative Reconfigurable Platform for Embedded System Design" The Workshop ZES’97, Rostock, 1997.

[GAU01] Lovic Gauthier, “Génération de système d’exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques“, Thèse de Doctorat INPG, Spécialité Microélectronique, laboratoire TIMA, 2001

[GHA03] Ferid Gharsalli, “Conception des interface logiciel-matériel pour l’intégration des mémoires globales dans les systems monopoces”, Thèse de doctorat, INPG, Spécialité Informatique, laboratoire TIMA, 2003

[GRA04] Grasset A., Rousseau F., Jerraya A.A. “Network Interface Generation for MPSOC: from Communication Service Requirement ti RTL Implementation”, 15th Rapid System Prototyping (RSP), 2004

[GRA04] Grasset A., Rousseau F., Jerraya A.A. “Network Interface Generation for MPSOC: from Communication Service Requirement ti RTL Implementation”, 15th Rapid System Prototyping (RSP), 2004

[KUR02] Kurshan R.P., Levin V., Minea M., Peled D., Yenigün H., “Combining Software and Hardware Verification Techniques”, Kluwer Academic Publisher, 2002

[KOR01] Kordon F., Munier I., Paviot-Adet E., Reged D.,“Formal Verification of Embedded Distributed Systems in a Prototyping Approach”, dans Monterey Workshop 2001 : on Engineering Automation for Software Intensive System Integration, 2001.

[LYO03] Damien Lyonnard, “Approche d’assemblage systématique d’élément d’interface pour la generation d’architectures multiprocesseurs”, Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2003

[MEN03] Mentor Emulation Division, “Accelerated SoC Verification Solutions” http://www.mentor.com/emulation, Mentor Graphics, 2003

[MES00] Mestdagh D.J.G., Isakson M.R., Odling P., Zipper VDSL: A Solution for Robust Duplex Communication over Telephone Lines, IEEE Communication Magazine, pp. 90-96, mai ,2000

[MOS96] Mosanya E., Goeke M., Linder J., Perrier J.Y., Rampogna F., Sanchez E. "Platform for Codesign and cosynthesis based on FPGA", 12th Rapid System Prototyping (RSP), Thessaloniki, 1996

[MPI95] “The Message Passing Interface (MPI) standard”, http://www-unix.mcs.anl.gov/mpi/, Version 1.1, 1995

[NET03] NetChip Tehnology, Inc., “Getting started with Netchip’s NET2280 PCI-RDK USB Host et USB Device Software”, NetChip Technology Inc., 2003

[NIC02] Eugenia G. N. Nicolescu, “Spécification et validation des systèmes hétérogènes embarqués” , Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2002

[PAV01] Pavesi M., Gemelli R., Ferloni M., Deroux-Dauphin P., Bazillz B., Moreau J-P., Moussa I, Sbarra G., Gesano A., Costo- loni A, Vacca P., “FlexBench™: Design Flow of a Novel Platform for Rapid Prototyping”, Design Automation and Test Europe, Munich, 2001

[PAV04] Yanick Paviot, “Implémentations mixtes logicielles/matérielles des services de communication pour l’exploration du partitionnement logiciel/matériel”, Thèse de doctorat, INPG, Spécialité Microélectronique, laboratoire TIMA, 2004

[POLIS] http://www-cad.eecs.berkeley.edu/~polis

[RAM01] Ramanathan A., Teisser R., McLaughin D., "Acquisition of Sensing Data on a Reconfigurable Platform", International Geosciences and Remote Sensing Symposium, Sidney, 2001

[SAS03a] Sasongko A, Baghdadi A., Rousseau F., Jerraya A.A., “*Prototyping of Embedded Applications to Configurable Platform Driven by Communication Constraints*”, 14th Rapid System Prototyping (RSP), San Diego, USA, 2003

[SAS03b] Sasongko A., Baghdadi A., Rousseau F., Jerraya A.A., “*Towards SoC Validation through Prototyping: A Systematic Approach Based on Configurable Platform*”, dans DAES volume 8 numero 2/3, pages : 115-173, Kluwer Academic Publisher, 2003

[SAS03c] Sasongko A, Baghdadi A., Rousseau F., Jerraya A.A., “SoC Validation trough Prototyping on Arm Integrator Platform”, in une journaux : “Information Quarterly”, ARM, 2003

[SAS03d] Sassatelli G, Torres L., Benoit P., Gil T., Robert M., Cambon G., Galy J., "Highly Scalable Dynamically Reconfiguable Ring-Architecture for DSP Application" 5th edition Sophia Antipolis Forumon Microelectronics, Sophia Antipolis, 2003.

[SAW96] Sanjay Sawant, “RTL Emulation: The next Leap in System Verification”, Design Automation Conference (DAC), Las Vegas, 1996

[STA94] Jorgen Staunstrup,”A Formal Approach to Hardware Design”, Kluwer Academic Publisher, 1994

[SYS99] Synopsys Inc., SystemC disponible à http://www.systemc.org/

[WEB00a] John G. Webster, “Modeling and Simulation”, dans “Wiley Encyclopedia of Electrical and Electronics Engineering”, volume 13, pages : 404-416, John Wiley and Son, 1999

[WED00b] John G. Webster, “Emulator”, dans “Wiley Encyclopedia of Electrical and Electronics Engineering”, volume 7, pages : 79-92, John Wiley and Son, 1999

[WEB00c] John G. Webster, “Design Verification and Fault Diagnosis in Manufacturing”, dans “Wiley Encyclopedia of Electrical and Electronics Engineering” volume 5, pages : 221-233, John Wiley and Son, 1999

[WEB00d] John G. Webster, “Rapid Prototyping System”, dans “Wiley Encyclopedia of Electrical and Electronics Engineering”, volume 18, pages : 234-241, John Wiley and Son, 1999

[WINSY] http://www.winsystems.com

[YOU04] Youssef M. W., Yoo S., Sasongko A., Paviot Y., Jerraya A.A., “Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study”, Design Automation Conference(DAC), San Diego, 2004

PUBLICATIONS

1. SASONGKO A, BAGHDADI A., ROUSSEAU F., JERRAYA A. A., "*Prototyping of Embedded Applications to Configurable Platform Driven by Communication Constraints*", dans "*Proceeding of Workshop on Rapid System Prototyping*", San Diego, Etats-unis, 2003

2. SASONGKO A, BAGHDADI A., ROUSSEAU F., JERRAYA A. A., *"Towards SoC Validation through Prototyping: A Systematic Approach Based on Configurable Platform"*, dans une revue internationale, "Design Automation of Embedded System Vol. 8 No2/3", Kluwer Academics Publisher, 2003

3. SASONGKO A, BAGHDADI A., ROUSSEAU F., JERRAYA A. A., *"SoC Validation trough Prototyping on Arm Integrator Platform"*, dans un journal, "*Information Quarterly*", ARM, 2003

4. YOUSSEF M. W., YOO S., SASONGKO A., PAVIOT Y., JERRAYA A. A., *"Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study"*, dans "The proceeding of 41st Design Automation Conference", San Diego, Etats-unis, 2004

RESUME

La technologie facilite l'intégration de nombreux composants sur une puce pour atteindre les performances et les besoins exigés par les applications. La tendance est à l'augmentation de la complexité et de l'hétérogénéité de tels systèmes, appelés systèmes monopuces.

Les systèmes monopuces sont sur un marché très concurrentiel, et l'arrivée rapide du produit sur le marché est très importante. De plus, le coût lié à la conception des parties matérielles et logicielles est très élevé. Détecter une erreur après fabrication entraîne un surcoût financier et de temps non acceptable. Ceci nous entraîne vers les deux problèmes traités dans ce travail de thèse : s'assurer que la système est correct avant sa fabrication et accélérer le processus de conception.

Après avoir évalué plusieurs techniques de vérification, nous pensons que le prototypage sur une plateforme reconfigurable est une solution adaptée pour les problèmes mentionnés. Ce prototypage permet de vérifier rigoureusement les systèmes grâce à une vitesse élevée, et de tester le système dans son environnement d'utilisation. Il accélère aussi la conception en permettent le développement de certaines couches logicielles avant que le système soit fini.

Pour obtenir rapidement un prototype à partir d'une description RTL d'une application, nous proposons un flot de prototypage basé sur une plateforme reconfigurable. Ce flot est composé de quatre étapes : allocation, configuration de la plateforme, adaptation de l'application, et génération du code.

Dans **l'allocation**, les concepteurs associent chaque partie de l'architecture à un nœud de prototypage de la plateforme. Ces associations indiquent sur quelles parties de la plateforme reconfigurable sont réalisées les parties de l'architecture de l'application. **La configuration** est la réorganisation de la plateforme reconfigurable. **L'adaptation** consiste à modifier l'application pour satisfaire aux caractéristiques de la plateforme reconfigurable. Cette étape est effectuée si la plateforme ne peut pas être configurée pour s'adapter aux besoins de l'application. Enfin, **la génération du code** est un processus standard tel que la compilation et l'édition de lien des logiciels, la synthèse logique, le placement sur FPGA, et le routage.

Ce flot a été validé en réalisant le prototypage des applications VDSL et DivX. La plateforme utilisée est une plateforme ARM Integrator avec une carte mère, quatre modules processeurs ARM, et d'un module FPGA communiquant à travers un bus AMBA AHB. Une expérience de co-émulation a également été réalisée pour explorer les difficultés et les avantages de cette technique. L'avantage principal est qu'on peut profiter de l'observabilité de la simulation et de la vitesse de l'émulation.

Ce travail de thèse montre que l'on peut obtenir rapidement un prototype en utilisant le flot propose sur une plateforme reconfigurable et aussi faciliter le développement des parties logicielles pour accélérer la conception. La configurabilité de plateforme de prototypage et l'intégration du flot de prototypage sur un flot de conception des systèmes restent des problématiques à approfondir.

ABSTRACT

The technology facilitates integration of many components onto a single chip to achieve performances and requirements needed by the application. The complexity and the heterogeneity of this system, called system-on-chip (SoC), tend to increase.

The market of SoC is very competitive, so early appearance on the market is very important. Furthermore, the cost chip fabrication is very high, therefore, detecting a bug after fabrication can cause unacceptable overhead. These facts bring us to two problems addressed in this thesis: assuring the correctness of the system and accelerating the design process.

After evaluating several verification techniques, we conclude that prototyping based on reconfigurable platform is a solution for the two problems mentioned. This prototyping allows us to verify rigorously the system since the speed which is very high. It allows us also to test the system in his operating environment. Furthermore, prototyping accelerate the design process by allowing development of several software layers before the chip fabrication.

To obtain quickly a prototype from RTL description of the application, we propose a prototyping flow based on reconfigurable platform. This flow consists of four steps: assignment, configuration, adaptation, and code generation.

In the **assignment** step, the designer associates each part of the architecture to the prototyping node of the prototyping platform. These associations indicate parts of the prototyping platform which will implement the components of the application. **Configuration** is reorganization of the reconfigurable platform. **Adaptation** consists of modify the application to satisfy constraints of the platform. This step is needed when the platform can not be configured to adapt the requirements of the application. Finally, the **code generation** is standard process such as compilation, logic synthesis, and placement and route.

This flow is validated by realizing two prototypes of application: VDSL and DivX encoder. In these experiments, we used ARM Integrator platform. This platform consists of a main board, four processor boards, an FPGA board. These boards communicate each others through bus AMBA-AHB. A co-emulation experiment is also performed using this platform for exploring the difficulties and the advantages of this technique. The main advantage is that we can obtain the observability of simulation while preserving the speed of emulation.

This PhD work shows that we can obtain a prototype using the proposed flow on a reconfigurable platform and also facilitate the development the software part to accelerate the design process. The configurability of the platform and the integration of the prototyping flow with design flow of the SoC are left as subjects to be treated. ISBN 2-84813-054-7